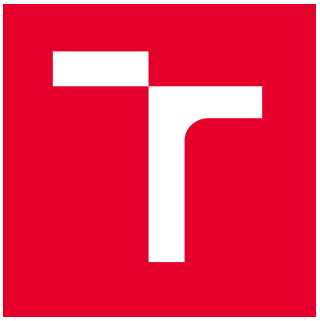
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2025 Karol Nový



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

MOBILNÍ APLIKACE PRO SPRÁVU ROBOTICKÝCH PAŽÍ

MOBILE APP FOR ROBOTIC ARM MANAGEMENT

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Karol Nový**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Karel Kuchař**

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Karol Nový ***ID:*** 247154

Ročník: 3 ***Akademický rok:*** 2024/25

NÁZEV TÉMATU:

Mobilní aplikace pro správu robotických paží

POKYNY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce je návrh, implementace a testování mobilní aplikace pro ovládání robotických paží Dobot pro OS Android. Mobilní aplikace bude umožňovat provádění základních úkonů podporovaných zařízením Dobot skrze uživatelské rozhraní, dále bude umožňovat vytvoření opakujícího cyklu (základní programování prováděné uživatelem aplikace skrze definované bloky). Mobilní aplikace bude dále schopna poskytovat informace o stavu robotických paží a poskytovat upozornění v případě definovaných stavů. Součástí bakalářské práce bude vytvoření scénáře pro demonstraci zranitelností mobilních aplikací a jejich dopady pro průmyslové sítě. V rámci teoretické části práce student provede rešerši vhodných nástrojů pro tvorbu mobilní aplikace a nástrojů pro vzdálené ovládání a monitoring průmyslových sítí. Výstupem práce bude návrh aplikace, její implementace a následné testování funkčnosti formou definovaných scénářů a dokumentace.

DOPORUČENÁ LITERATURA:

[1] BALAPOUR, Ali; NIKKHAH, Hamid Reza a SABHERWAL, Rajiv, 2020. Mobile application security: Role of perceived privacy as the predictor of security perceptions. Online. International Journal of Information Management. Roč. 52. ISSN 02684012. Dostupné z: https://doi.org/10.1016/j.ijinfomgt.2019.102063

[2] KNAPP, Eric, 2024. Industrial Network Security: Securing Critical Infrastructure Networks for Smart Grid, SCADA, and Other Industrial Control Systems. Online. 3. Syngress. ISBN 978-0443137372

Termín zadání: 10.2.2025 ***Termín odevzdání:*** 3.6.2025

Vedoucí práce: Ing. Karel Kuchař

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce se věnuje využití průmyslových sítí, konkrétně protokolu Modbus/TCP, v mobilních aplikacích Flutter. V teoretické části jsou popsány průmyslové sítě a porovnány síťové protokoly. Dále se práce zabývá vývojem mobilních aplikací, porovnáváním aktuálně rozšířených frameworků a popisováním nástrojů pro správu zdrojového kódu. V praktické části byly navrženy dvě realizace řízení robotické paže pomocí mobilní aplikace. První se zabývá komunikací mezi mobilní aplikací a robotickou paží pomocí HTTP protokolu a skriptů v Pythonu. Druhá se věnuje generování Modbus protokolu přímo z mobilní aplikace. Dále je v neposlední řadě vyzdvižena zranitelnost mobilních aplikací. Nakonec bylo provedeno testování řízení robotické paže, možnosti uživatelského programování a zachytávání komunikace mezi mobilní aplikací a robotickou paží.

KLÍČOVÁ SLOVA

Modbus, mobilní aplikace, Průmyslové sítě (OT), Flutter

ABSTRACT

The bachelor's thesis is devoted to the use of industrial networks, specifically the Modbus/TCP protocol, in mobile applications. The theoretical part describes industrial networks and compares network protocols. The work also deals with the development of mobile applications, comparing currently widespread frameworks and describing tools for source code management. In the practical part, two implementations of controlling a robotic arm using a mobile application were designed. The first deals with communication between a mobile application and a robotic arm using the HTTP protocol and scripts in Python. The second is devoted to generating a Modbus protocol directly from a mobile application. Last but not least, the vulnerability of mobile applications is highlighted. Finally, testing of the robotic arm control, user programming options and interception of communication between the mobile application and the robotic arm was carried out.

KEYWORDS

Industrial Networks (OT), Modbus, mobile applications, Flutter

NOVÝ, Karol. *Mobilní aplikace pro správu robotických paží*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2025. Vedoucí práce: Ing. Karel Kuchař

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Karol Nový
VUT ID autora:	247154
Typ práce:	Bakalářská práce
Akademický rok:	2024/25
Téma závěrečné práce:	Mobilní aplikace pro správu robotických paží

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Karlu Kuchaři za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

V dnešní době se mobilní aplikace dostávají čím dál více do popředí. Populární jsou zejména díky jejich multifunkčnosti, či lehké manipulaci. Jak vyjadřuje jejich název, tak se používají na mobilním zařízení, díky čemuž jsou i lehce přenosné. Stejným tempem se rozvíjí i nástroje pro vytváření mobilních aplikací, což taktéž velmi přispívá ke stále rostoucí využitelnosti. Dále se sem řadí integrovaná vývojová prostředí pro správu a vývoj mobilních aplikací. Aktuálně se mobilní aplikace hojně vyskytují v obchodním, či herním průmyslu. Také se uplatňují jako sociální sítě, nebo jako prostředník mezi restaurací a zákazníkem. Naopak nejsou velmi používané ve strojírenském, elektrotechnickém či automatizačním průmyslu.

Tato semestrální práce se zabývá možností využití mobilních aplikací v robotickém průmyslu, konkrétně řízením robotické paže pomocí mobilní aplikace v mobilním telefonu. Hlavním tématem je zde popis a průzkum možností komunikace mezi mobilními aplikacemi a průmyslovými protokoly. První část práce je zaměřena na popis průmyslových sítí a jejich náležitostí, jako jsou komunikační modely ISO/OSI a TCP/IP. Dále se tato část zabývá rešerší nejpoužívanějších průmyslových protokolů, mezi které spadají protokoly Modbus či Profibus, a jejich porovnáním. Druhá část se věnuje struktuře, možnostem a vývoji mobilních aplikací. Dále rešerší nejpoužívanějších nástrojů pro vývoj těchto aplikací — frameworky, integrovaná vývojová prostředí, emulátory — a jejich možnostem pro komunikaci s průmyslovými protokoly.

Na základě předešlých a získaných poznatků je v další části navržena struktura výsledné komunikace mezi mobilní aplikací vytvořenou pomocí frameworku Flutter a robotickými pažemi v podobě blokového schématu. Dále je podrobně popsán každý blok komunikace – uživatel, aplikace Flutter, server Flask, skripty, Raspberry Pi, robotická paže – který je nezbytnou částí pro realizaci. Ve zbylých částech této práce je zhotoven, ve webové aplikaci PenPot, vizuální a funkční návrh mobilní aplikace a zdrojový kód pro realizaci a spuštění této aplikace na mobilním zařízení. V neposlední řadě je zde zmíněn server naprogramovaný ve frameworku Flask, který zajišťuje komunikaci mezi mobilní aplikací a mikropočítačem Raspberry Pi, jenž řeší ovládání samotné robotické paže.

V rámci bakalářské práce je vyvíjena snaha o přidání nových funkcí do mobilní aplikace, jako je možnost uživatelského programování a podrobná statistika aktuálních stavů robotické paže. Dále se práce zabývá návrhem kvalitnějšího designu a využitím pokročilých animací, které Flutter nabízí. Velkou část zde taktéž zaujímá průzkum a testování balíčků, vytvořených pro Flutter, které nabízejí možnost generování protokolu Modbus, což vede k rychlejší a efektivnější komunikaci s robotickou paží.

1 Průmyslové sítě

Průmyslové sítě se v této době využívají v mnoha průmyslových oblastech včetně výroby elektřiny, zpracování potravin a nápojů, dopravy, či distribuci vody. Téměř v každé situaci, která to vyžaduje, je instalována v nějaké formě průmyslová řídící síť. Každé odvětví představuje svůj vlastní soubor mírně odlišných, ale obecně podobných požadavků[1][2].

Definice

Průmyslové sítě ze své podstaty využívají geografické rozložení fyzických měření I/O a senzorů nebo funkční rozložení aplikací. Princip většiny průmyslových sítí spočívá v sériovém přenosu bitů, které nesou informaci. Sériový přenos dat má tu výhodu, že vyžaduje pouze omezený počet vodičů pro výměnu dat mezi zařízeními. Jelikož průmyslové sítě pracují s několika zařízeními na stejné lince, je o to snadnější přidat další zařízení ke stávajícím systémům[1][2].
Aby vše správně fungovalo, tak tato síť musí definovat sadu pravidel. Tyto pravidla udávají komunikační protokoly, které řídí způsob toku informací v síti daného zařízení. S vylepšenými komunikačními protokoly je možné zajistit lepší ochranu dat, zkrátit čas pro přenos, zaručit synchronizaci času a dokonce, u některých aplikací, deterministickou odezvu v reálném čase[1][2].

Složení

Průmyslové sítě se skládají ze specializovaných komponent a aplikací, jako jsou programovatelné logické automaty (PLC), dispečerské řízení a sběr dat (SCADA) a distribuované řídící systémy (DCS). To je komunikace uvnitř a mezi těmito složkami a systémy, kterými se průmyslové sítě primárně zabývají[1][2].
Obecně jsou informační technologie (IT) zařízení navržena pro sběr a odesílání dat na servery.Jsou přímočaré a původně určené pro jednosměrný přenos dat. Naproti tomu předchůdce těchto zařízení, tedy operační nebo provozní technologie (OT), jako jsou PLC, se zaměřovala na úkoly prováděné na místě nebo prostřednictvím programování na počítači (PC), s omezenými komunikačními schopnostmi[1][2].
Komunikace mezi těmito zařízeními se zlepšila zavedením nových komunikačních protokolů jako Ethernet/IP, DeviceNet a ControlNet. Tyto protokoly umožňovaly OT zařízením vyměňovat si data v reálném čase a tedy efektivněji spolupracovat. V důsledku toho se IT a OT začaly slučovat, což vedlo k lepší viditelnosti a efektivitě různých operací[1][2][4].

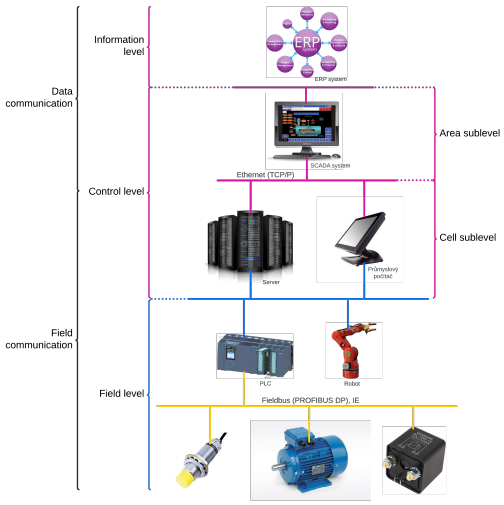
- **PLC** jsou specializované, počítačově založené, polovodičové, elektronické zařízení, která tvoří jádro průmyslového řízení sítí. Někdy označované jako programovatelné ovladače (PC), avšak PLC se používá, aby se předešlo záměně se zkratkou pro osobní počítač[1][2][6].
  Programy pro PLC jsou obvykle psány na počítači a mnoho výrobců vytvořilo tzv. vývojová prostředí, která napomáhají při vývoji těchto programů. Skutečné programování PLC se provádí prostřednictvím softwarového programování, buď pomocí fyzického připojení k vyhrazenému portu na zařízení a nebo prostřednictvím sítě, ke které je zařízení (PLC) připojeno[1][2][6].
- **SCADA** systém je čistě softwarová část používaná na úrovni nad ovládacím hardwarem v rámci hierarchie průmyslových sítí. SCADA systémy, jako takové, neprovádějí žádnou kontrolu. Jejich hlavním úkolem je sběr dat a reprezentace rozhraní člověk-stroj (HMI). Dále umožňují také příkazy na vyšší úrovni, které mají být odeslány do řídícího hardwaru, jako je například pokyn ke spuštění motoru nebo změně nastavené hodnoty[1][2][6].
- **DCS** se svou funkcí podobá SCADA systému. Rozdíl mezi nimi je často nepatrný, zejména s pokrokem v technologii, který umožňuje překrývání funkcí každého z nich. Klíčový rozdíl mezi nimi je, že DCS jsou řízeny spíše procesem, než událostmi. Jejich hlavní a konkrétní rozdíly jsou zobrazeny v tab. 1.1[1][2][6].

Tab. 1.1: DCS a SCADA. Zdroj: [1]

DCS	**SCADA**
Řízené procesem	Řízené událostmi
Malé zeměpisné oblasti	Velké zeměpisné oblasti
Vhodný pro velké integrované systémy, jako je například chemické zpracování nebo výroba elektřiny	Vhodný pro více nezávislých systémů, jako je diskrétní výroba a distribuce
Dobrá kvalita dat a spolehlivost médií	Nižší kvalita dat a spolehlivost médií
Výkonný řídící hardware s uzavřenou smyčkou	Výkonově účinný hardware, často zaměřený na detekci binárních signálů

1.1 Hierarchické úrovně

Systémy průmyslových komunikačních sítí mohou být velmi složité a jsou tedy obvykle strukturovány do několika hierarchických úrovní. Každá hierarchická úroveň má svou komunikační úroveň, která klade na komunikační síť různé požadavky. Obrázek 1.1 ukazuje příklad těchto hierarchických úrovní [3].



Obr. 1.1: Hierarchie průmyslových sítí

- **Nejnižší úroveň (Field level)** – Tato úroveň zahrnuje provozní zařízení, jako jsou akční členy a senzory. Úkolem těchto zařízení je průmyslový proces. Průmyslové sítě na této úrovni jsou velkou kategorií, která se vyznačuje vlastnostmi, jako je velikost zprávy a doba odezvy.[3].
- **Úroveň kontroly (Control level)** – Na této úrovni se informační tok skládá především z načítání dat, programů a parametrů. Tuto úroveň lze rozdělit na dvě úrovně: buněčnou a plošnou. Buněčná úroveň se zaměřuje na určitou část procesu, která zahrnuje obvykle jednotlivé výrobní buňky, které obsahují skupinu zařízení, jako jsou stroje nebo roboti, kteří jsou určeni pro konkrétní operaci nebo úkol. Typickými příklady pro tuto úroveň jsou servery, nebo průmyslové počítače, které mohou sloužit jako řídící jednotka pro konkrétní buňku. Zajišťují monitorování a řízení provozu strojů a zařízení v rámci této buňky a také mohou sbírat a analyzovat data z různých senzorů. Plošná úroveň se skládá z buněk spojených do skupin. Tato úroveň se týká širší oblasti výrobního procesu jako je továrna nebo výrobní závod. Zaměřuje se na správu a koordinaci operací mezi různými výrobními buňkami, monitoruje jejich stav a zajišťuje efektivní výrobu a tok informací na úrovni celého výrobního procesu. Mezi typické příklady patří SCADA systémy, HMI a další řídící systémy, které dokážou monitorovat a řídit operace v rámci více buněk v určité oblasti. Mimo jiné se často používá Ethernet s TCP/IP jako sběrnice řadiče pro připojení řídících zařízení a počítačů vyšší úrovně [3].
- **Informační úroveň (Information level)** – Tato úroveň slouží jako vrstva pro sběr, analýzu a správu dat z různých zdrojů. Shromažďuje informace o řízení z různých úrovní, včetně plošných a řídí celý síťový systém. Je důležitá pro rozhodování a optimalizaci procesů. Existují zde rozsáhlé sítě, jako například

ethernetové WAN, systémy MES nebo ERP [3].

1.2 Komunikační modely

Průmyslové modely specifikují sadu pravidel pro daný typ komunikace. Mezi dva nejdůležitější patří ISO/OSI a TCP/IP [2].

ISO/OSI

Model ISO/OSI (Open System Interconnection) byl vytvořen organizací ISO (International Standards Organisation), která poskytla společný standard ISO 7498 pro všechny vrstvy počítačových sítí.
V tomto modelu se sada protokolů rozděluje na 7 tzv. OSI vrstev, které fungují následovně:

- každá vrstva podporuje protokol nezávisle na ostatních vrstvách
- každá vrstva poskytuje bezprostředně služby vrstvě, která je nad ní
- každá vrstva vyžaduje bezprostředně služby vrstvy, která je pod ní

Obecně 1. vrstva popisuje fyzické komunikační médium a 7. vrstva poskytuje služby uživateli nebo aplikaci [2].
Obecný princip spočívá v tom, že při komunikaci volá uživatel sítě přes služby 7. vrstvy program. Tato vrstva formátuje a obohacuje data, která jí program poskytl dle svého protokolu a odešle je do vrstvy pod ní, pokud toho je potřeba. Další vrstvy dále formátují data dle jejich protokolů a nakonec jsou výsledná data odeslána do fyzického média a přijata jiným síťovým uzlem. Reverzním principem se poté data dostanou zpět a končí v programu korespondenta [2].
7-vrstvý model OSI, který je možný vidět na obr. 1.2 byl implementován několika výrobci, ale nikdy nebyl komerčně úspěšný, jelikož trh preferoval 4-vrstvový TCP/IP model, který je snadněji pochopitelný a lépe použitelný [2].

TCP/IP

Model TCP/IP se skládá pouze z fyzické, síťové, transportní a aplikační vrstvy, jak můžete vidět také na obr. 1.2. Tento model je v tuto dobu nejvíce používaný v rámci internetu a slouží jako základ pro Ethernet v reálném čase. Implementace tzv. "Real-time" Ethernetu jsou založeny na tomto čtyřvrstvém modelu TCP/IP. Požadavky v reálném čase splňuje tento model prostřednictvím jednoho ze tří přístupů. Společný pro všechny je použití ethernetové kabeláže a TCP nebo UDP protokolů, které zajišťují přenos dat, respektive paketů či rámců, viz. obr. 1.3, mezi zařízeními v síti [2].

- **Fyzická vrstva** – popisuje fyzické vlastnosti komunikace jako je typ běžně používaného média (elektrické kabely, optická vlákna nebo rádiové spojení) a všechny související detaily, jako konektory, typ kódování a modulace, úrovně signálu, nebo vlnové délky [2].
- **Spojová vrstva** – určuje řízení přístupu k médiím a způsob přenosu datových paketů na fyzické vrstvě, zejména v rámcové struktuře (specifické sekvence bitů na začátku a na konci paketů). Například záhlaví ethernetových rámců obsahuje pole označující přístroj a síť, do které má paket jít [2].
- **Síťová vrstva** – řeší problémy přenosu datových paketů v rámci jedné sítě. Přenáší data ze zdrojové sítě na cílovou. Obecně to znamená, že pakety jsou směrovány přes internet. V sadě internetových protokolů přenáší IP pakety ze zdroje do cíle kdekoliv na světě. Princip IP adresování spočívá v zajištění jedinečnosti každé IP adresy, tedy každá stanice má svou vlastní, jedinečnou IP adresu. Protokol IP zahrnuje i další protokoly, jako je ICMP, používaný pro přenos zpráv o chybách, nebo IGMP, který spravuje multicastová data [2].
- **Transportní vrstva** – protokoly transportní vrstvy mohou řešit problémy, jako je spolehlivost výměny dat, automatické přizpůsobení kapacity sítě a řízení datového toku. Zajišťuje také, že data dorazí ve správném pořadí. V sadě protokolů TCP/IP transportní protokoly určují, do které aplikace má být každý datový paket doručen. Například TCP je spojově orientovaný a zajišťuje všechny potřebné parametry, o kterých už byla zmínka výše. V případě ztráty a odstranění duplicitních dat vytvoří opakovaný přenos. Snaží se doručit všechna data správně a v pořádku, což je jeho účel a hlavní výhoda oproti UDP. UDP je naopak jednoduchý a rychlý protokol, který je avšak nespolehlivý, jelikož nekontroluje, zda se pakety dostaly do správného cíle, ani to, zda dorazily v pořádku. Pokud tedy aplikace vyžaduje záruky, že se data přenesou správně a všechna, tak si to buď musí zajistit sama, nebo použít TCP protokol. UDP se obvykle používá pro vysílací aplikace, jako jsou Global Data nebo multimediální aplikace (audio, či video) kde není dostatek času na ří-

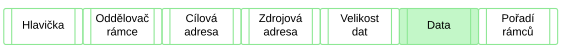
zení opakovaného přenosu a řazení paketů, jako u TCP. TCP protokol využívá například Modbus TCP [2].

- **Aplikační vrstva** – většina funkcí síťových aplikací je umístěna v aplikační vrstvě. Patří mezi ně například HTTP pro web, FTP pro přenos souborů, SMTP pro zasílání zpráv, SSH pro zabezpečené vzdálené připojení, DNS pro odpovídající IP adresy a jména.
  Aplikace obecně fungují pod TCP nebo UDP a jsou obvykle propojené do známého portu [2].
  - HTTP na port TCP 80 nebo 8080
  - Modbus na port 502
  - SMTP na port 25

Obr. 1.2: ISO/OSI a TCP/IP. Zdroj: [2]

Rámec

Rámec, taktéž paket, zobrazený na obr. 1.3, je soubor dat odeslaných sítí v jediném bloku. Každý rámec má stejné základní rozložení a obsahuje řídící informace, jako jsou adresy pracovních stanic, synchronizační znaky, chybové a kontrolní hodnoty a různá velká množství dat [2].



Obr. 1.3: Standardní formát rámce. Zdroj: [2]

1.3 Průmyslové protokoly

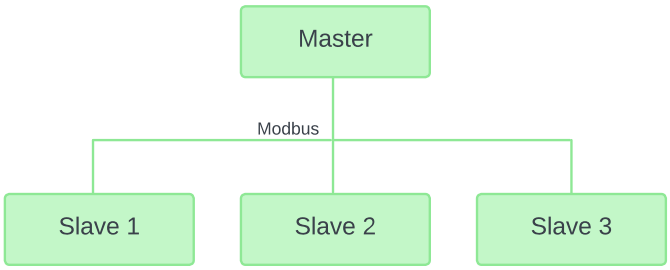
Průmyslové protokoly jsou "Real-time" komunikační protokoly, vyvinuté k propojení systémů, rozhraní a nástrojů, tvořících průmyslový řídící systém. Většina byla navržena pro sériovou komunikaci přes RS-232, RS-485 nebo jiná sériová připojení, avšak v dnešní době už jsou vyvinuty tak, aby fungovaly prostřednictvím ethernetové sítě pomocí směrovacích protokolů, jako je například TCP/IP [5][6].
Mezi nejvíce používané průmyslové protokoly patří **Modbus**, a jeho verze **Modbus RTU**, **Modbus ASCII** a **Modbus TCP**. Dále **PROFINET/PROFIBUS** nebo **EtherNet/IP** [5][6].

1.3.1 Modbus

Modbus je komunikační protokol pro komunikaci s PLC. Díky tomu, že je to otevřený, jednoduše implementovaný protokol, se stal jedním z nejpoužívanějších protokolů v průmyslu [7][8].

Sériový linkový protokol

Protokol sériové linky Modbus pracuje v režimu master-slave viz. obr. 1.4 a využívá poloduplexní přenos dat. Režim master-slave znamená, že komunikace nemůže probíhat mezi podřízenými jednotkami (slave) navzájem. Teprve až hlavní jednotka (master) odešle data na slave (odešle požadavek), tak slave přijme data a odpoví dalšími daty, jako je znázorněno na obr. 1.5 [7][8].
Standard RS485 obvykle vyžaduje dvě linky a v určitou dobu, zde probíhá komunikace mezi jedním masterem a více slave zařízeními [7][8]. Při režimu **master-slave**



Obr. 1.4: Komunikační schéma Modbus. Zdroj: [8]

je typicky na sběrnici jeden master a více slave zařízení. Každý slave má svůj jedinečný identifikátor (ID), díky kterému master komunikuje se slave zařízením, pro přenos dat [7][8].

Obr. 1.5: Master-slave komunikace. Zdroj: [8]

Komunikační proces

Jak již bylo řečeno, v Modbus komunikaci může odeslat požadavek jedno či více zařízení. Slave přijímají data od master a nějakým způsobem reagují. Slave jsou periferní zařízení, jako je snímač teploty nebo vlhkosti, detektor kouře, či snímač hladiny vody [7][8].
Slave zpracuje informace a pošlou zpět svá data na master. Slave nikdy nezahajují komunikaci, mohou pouze odpovídat na zprávy zaslané z master zařízení [7][8].
Kromě toho není v komunikačním procesu žádný vestavěný mechanismus, který by určil, zda je, nebo není, zařízení zaneprázdněno. Pokud tedy například master odešle příkaz na slave, ale slave jej buď neobdržel, nebo je zaneprázdněn jinými příkazy, tak nemůže masteru odpovědět. Je to zapříčiněno tím, že sběrnice RS485 je výhradně zodpovědná za přenos dat a postrádá tedy další mechanismy zpracování. Pro určení správného příjmu dat je tedy nutné použít některé softwarové přístupy [7][8].

Verze protokolu Modbus

Mezi hlavní verze protokolu Modbus patří **RTU** (Remote Terminal Unit), **ASCII** a **TCP** [7][8].

- **Modbus RTU** je režim, který často používá RS-485 jako fyzickou vrstvu a typicky využívá sériový port pro přenos datových zpráv. Datové zprávy jsou předávány v binárním formátu [7][8].
- **Modbus ASCII** je režim, při kterém jsou zprávy přenášeny pomocí ASCII znaků. Formát ASCII používá vertikální kontrolu redundance (VRC). Zprávy zařínají dvojtečkou („:“) a končí sekvencí návratu (CR/LF) [7][8].
- **Modbus TCP/IP** nebo **Modbus TCP** je varianta Modbusu, která používá ke komunikaci síť TCP/IP a připojuje se přes port 502. V Modbus TCP není výpočet kontrolního součtu zpráv vyžadován, jelikož vrstva Ethernetu již tuto integritu dat implementuje (CRC32) [7][8].

Obecně při použití sériové komunikace pro protokol Modbus je možné si vybrat mezi režimy RTU nebo ASCII. Oba tyto režimy mají definované struktury zpráv

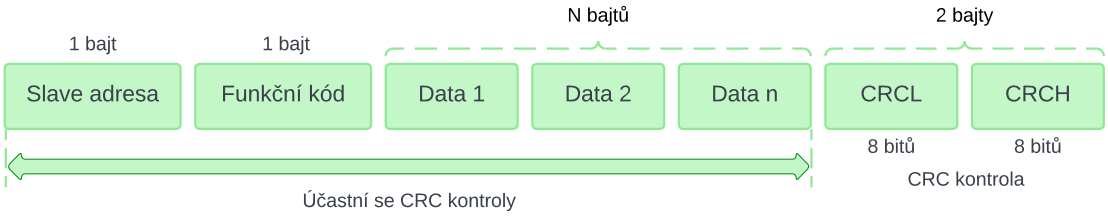
a dat, formáty příkazů a vyžadují ověření dat. Režim ASCII využívá konkrétně Longitudinal Redundancy Check (LRC) pro ověření dat, zatímco RTU používá 16-ti bitovou Cyclic Redundancy Check (CRC) [7][8].
Při přenosu Modbusu přes Ethernet pomocí TCP/IP jsou data zapouzdřena do TCP paketů. V tomto režimu není ověření nutné, jelikož TCP je spolehlivý protokol, který již tyto kontroly poskytuje [7][8].

Modbus RTU

Modbus RTU je kompaktní formát, který představuje data v hexadecimální soustavě. V RTU formátu je každý příkaz následován kontrolním součtem typu CRC. Pokud například potřebujeme odeslat číslo 10, v režimu RTU bychom poslali 0x0A. Přenos dat po sběrnici by byl tedy reprezentován jako 0000 1010 [7][8].

Formát rámce u Modbus RTU

Zpráva zde viz. obr. 1.6 odkazuje na rámec dat, který master posílá na slave. Zahrnuje adresu slave, zařízení, operaci, kterou chce master provést, kontrolní součet a další potřebné informace. [7][6].



Obr. 1.6: Formát rámce RTU. Zdroj: [8]

- **Slave adresa**: Každé slave zařízení má svou jedinečnou adresu, která zabírá 1 bajt s rozsahem 0-255. Platný rozsah pro slave adresy je 1-247, protože 255 je vhrazeno pro broadcast adresu (vysílání zprávy všem podřízeným zařízením) [7][8].
- **Funkční kód** označuje účel instrukce. Je možné například použít kódy funkcí ze slave zařízení k dotazování, nebo úpravu slave dat [7][8].
- **Data**: Datová část se liší v závisloti na funkčním kódu. Pokud je například funkční kód použit pro dotazování na data ze slave zařízení, tak data mohou zahrnovat adresu, nebo počet bajtů ke čtení [7][8].
- **CRC kontrola**: Při přenosu dat může docházet k chybám. Kontrolní součet CRC nebo LRC se tedy používá k ověření správnosti přijatých dat [7][8].

Funkční kódy

Modbus protokol definuje více funkcí a tedy pro jejich pohodlné využití je každé funkci přiřazen funkční kód, který funguje jako identifikátor. Samotný protokol specifikuje přes dvacet funkčních kódů, ale běžně se jich používá 8, pro čtení a zápis do paměťových oblastí viz.obr. 1.2 [7][8].

Tab. 1.2: Nejběžnější funkční kódy RTU. Zdroj: [8]

Funkční kód	Popis funkce
01H	číst výstupní cívku
02H	číst vstupní cívku
03H	číst zadržovací registr
04H	číst vstupní registr
05H	zápis do jedné cívky
06H	zápis do jednoho registru
0FH	zápis do více cívek
10H	zápis do více regitrů

Výhody protokolu Modbus

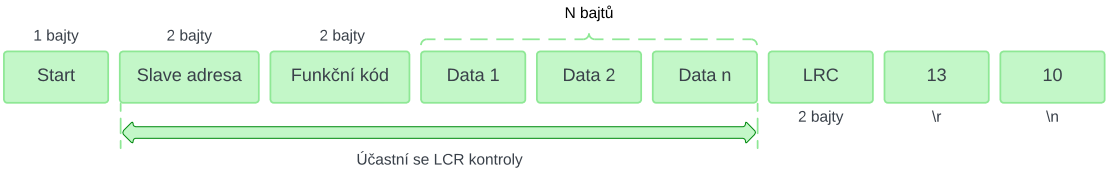
- Standard protokolu je otevřený, veřejně dostupný a nemá žádné požadavky na autorská práva.
- Protokol podporuje více elektrických rozhraní, včetně RS232, RS485, TCP/IP a může být také přenášen přes různá média, jako jsou kroucené dvojlinky, optika, infračervené a bezdrátové připojení [7][8].
- Formát zpráv paketů je jednoduchý, kompaktní a snadno pochopitelný. Je uživatelsky přívětivý a pro výrobce snadný na vývoj a integraci, což jsou důležité vlastnosti při vytváření průmyslových řídících sítí [7][8].

Modbus ASCII

Modbus ASCII je formát, kde jsou data vyjádřena pomocí kódů ASCII a každý bajt je přenášen jako dva ASCII znaky. Tento formát používá, pro kontrolu správnosti přijatých dat, kontrolní součet LRC [8].

Formát rámce u Modbus ASCII

V Modbus ASCII jsou počáteční a koncové znaky (CR LF), které slouží jako indikátory pro začátek a konec datového rámce. Na druhou stranu Modbus RTU takové značky nemá, jelikož spoléhá na časové intervaly pro určení začátku a konce rámce viz.obr.1.7 [8].



Obr. 1.7: Formát rámce ASCII. Zdroj: [8]

- **Adresa**: Představuje adresu slave zařízení.
- **LRC kontrola**: Narozdíl od Modbus RTU, je zde využit LRC pro kontrolu chyb. LRC se vypočítává pes ASCII reprezentaci adres, funkcí a datových bajtů [8].
- **Terminátor**: Rámec zprávy, který je ukončen znaky \r (návrat na začátek řádku) a \n (přechod na nový řádek), které označují konec zprávy [8].

Modbus TCP

Tato specifikace Modbus protokolu definuje vložení Modbus paketů do TCP rámců, přiřazení portu 502 k naslouchání a příjmu dat. Modbus TCP rámec obsahuje obvyklé hlavičky IP a TCP, následované datovou jednotkou aplikačního protokolu specifickým pro Modbus TCP (APDU), která se skládá ze dvou částí:

- **hlavička MBAP**, která je zobrazena v tab. 1.3
- **užitečná data** (PDU)

Obr. 1.8 ukazuje pole Modbus TCP APDU a jeho velikost. [9][10].
Užitečná data se liší velikostí, ale jsou omezena na 253 bajtů, aby byla zachována kompatibilita s Modbus sériovým protokolem[8][9][10].
Záhlaví MBAP se skládá ze čtyř polí:

- **Identifikátor transakce**, který se používá k párování transakcí, což je realizováno tím, že více zpráv může být přenášeno na stejném TCP spojení bez čekání na odpověď [9][10].
- **Identifikátor protokolu**, který je vždy 0 pro Modbus [9][10].
- **Délka pole**, která poskytuje délku zbývajících polí MBAP záhlaví a PDU [9][10].
- **Identifikátor jednotky**, který se používá k identifikaci umístěného PLC v síti bez TCP. Toto pole se tedy používá pro sériové přemostění, když jsou Modbus TCP a sériový Modbus ve stejné síti.
  Výchozí hodnoty pro toto pole jsou 00 nebo FF, což znamená, že by měl být ignorována opakován v odpovědi [9][10].

Modbus definuje tři typy PDU, tedy požadavek, odpověď a odpověď na požadavek [9][10].
PDU se skládá ze dvou polí:

- **Funkční kód**, který definuje typ PDU. Existují tři kategorie, tedy veřejné (standardní), definované uživatelem (funkce specifické pro dodavatele)a rezervované (využívané staršími produkty). Kontrétně jsou to kódy veřejných funkcí 1-8, 11-12, 15-17, 20-24 a 43 [9][10].
- **Data**

Obr. 1.8: Formát rámce TCP. Zdroj: [9]

Tab. 1.3: MBAP hlavička. Zdroj: [5]

Oblast	Délka	Popis	Klientský počítač	Server
Identifikátor transakce	2 bajty	Identifikace transakcí požadavek/odpověď	start klienta	server překopíroval požadavek
Identifikátor protokolu	2 bajty	0 = Modbus protokol	start klienta	server překopíroval požadavek
Délka pole	2 bajty	Počet bajtů, které následují	start klienta (požadavek)	start serveru (odpověď)
Identifikátor jednotky	1 bajt	Identifikace vzdálených slave, připojených na sériovou linku nebo jinou sběrnici	start klienta	server překopíroval požadavek

Zabezpečení

Zabezpečení Modbusu bylo široce studováno pomocí testování [11], formálních metod [12] nebo přístupů, které kombinují obě techniky [13].
Ve svém nativním výrazu, Modbus postrádá veškeré bezpečnostní atributy, což znamená, že nemá žádné ověřování, šifrování, integritu, což patří mezi hlavní bezpečnostní problémy tohoto protokolu. Tento nedostatek je, do značné míry, motivován skutečností, že Modbus byl navržen pro provoz v systémech s izolovanými sítěmi [9].

TLS

Protokol Transport Layer Security (TLS) je nejrozšířenějším protokolem pro zabezpečení síťového provozu. Poskytuje víceúčelové bezpečnostní služby, jako je vzájemná autentizace, důvěryhodnost a integrita dat, generování a distribuce klíčů, úprava bezpečnostních parametrů a podobně [14].

Modbus SECURITY

Tento nový protokol poskytuje robustní ochranu díky kombinaci zabezpečení Transport Layer Security (TLS) a tradičního protokolu Modbus. TLS byl vybrán, jelikož jde o dobře známou a široce přijímanou normu [15].
Princip vzpočívá v tom, že TLS zapouzdří pakety Modbus, aby bylo zajištěno ověřování a ochrana integrity zpráv. Využívá digitálních certifikátů X.509v3 pro autentizaci serveru a klienta. Protokol taktéž podporuje přenos informací o zařízení přístupu na základě rolí pomocí X.509v3 prodloužení pro autorizaci klientského požadavku. Na rozdíl od klasického Modbus protokolu – který využívá 502 – Modbus Security používá port 802 [15].

1.3.2 Profibus

PROFIBUS Medium Access Control (MAC) protokol používá předávání tokenů k udělení sběrnici přístup k master jednotkám. Po obdržení tokenu je PROFIBUS master schopen zpracovávat transakce během doby držení tohoto tokenu (TTH), která závisí na rozdílu času rotace cílového tokenu (TTR) a skutečného času tokenu (TRR) [16][17][18].
Transakce se skládá z rámce požadavku od master jednotky a souvisejícího potvrzovacího rámce, nebo odpovědi od slave jednotky. Odpověď z slave musí dorazit master jednotce před vypršením platnosti času slotu (TSL), což je hlavní parametr [16][17][18].
Aby byl zachován logický kruh, tak PROFIBUS poskytuje decentralizovaný mechanismus údržby kruhu. Každá master jednotka poskytuje dvě tabulky:

- **Seznam mezer** (GAPL)
- **Seznam aktivních stanic** (LAS)
- a může popřípadě udržovat aktuální seznam (LL).

GAPL se skládá z rozsahu adres z „Adresa stanice“ do „Další adresa stanice“. Pokaždé, když časovač aktualizace mezery (TGUD) v master jednotce vyprší, spustí se kontrola adres v jeho GAPL. Tato akce je, při každé návštěvě tokenu, provedena dotazem nejvýše jedné master jednotky. Pokud nový master odpoví, pak žádající master předá token tomuto novému a aktualizuje svou adresu na „Další stanice“.

V opačném případě žádající master pokračuje ve své činnosti [16][17][18].
LAS je seznam všech master jednotek v logickém kruhu a **LL** obsahuje všechny aktivní stanice, tedy obě master jednotky a slave jednotky [16][17][18].

Profibus DP

PROFIBUS DP je koncipován pro rychlou výměnu dat na úrovni senzor-akční člen a je integrován v programovatelné logice pomocí regulátorů tak, aby vyhovoval požadavkům budoucího vývoje. Lze zde uvést například PLC, z důvodu rychlého sériového spojení mezi vstupním a výstupním zařízením. Výměna dat mezi těmito zařízeními většinou probíhá cyklicky. Master čte vstupní informace od slave a zapisuje výstupní informace slave jednotkám. Distribuovaná periferie (OP) umožňuje používat velké množství digitálních a analogových vstupních/výstupních modulů a další zařízení distribuovaným způsobem, tedy lokálně [16][17][18].
Je optimalizován pro vysokou rychlost a levné připojení a je určen speciálně pro komunikaci mezi řídícími systémy automatizace[16][17][18].

Profibus PA

V zásadě je Profibus PA kompatibilní rozšíření Profibus DP. Se sběrnicovým mechanismem (nebo technologií přenosu dat), který byl speciálně vybrán pro plnění požadavků ve zpracovatelském průmyslu, jako je například vnitřní bezpečnost a interoperabilita systémů. Měřící převodníky a akční členy mohou spolu komunikovat pomocí centrály a zároveň být její pomocí napájeny. Tyto vysílače a akční členy mohou být také umístěny v nebezpečných oblastech a mohou se nacházet ve značné vzdálenosti od centrálního automatizačního systému[16][17][18].
Profibus PA je speciálně navržen pro automatizační procesy. Umožňuje senzorům a akčním členům se připojit na jednu sběrnici i v neobvyklých oblastech, jako jsou doly, chemické závody nebo podmořská těžba ropy.[16][17][18].
Také povoluje datovou komunikaci a napájení přes dvoudrátovou sběrnici pomocí technologie podle mezinárodního standardu IEC 1158-2[16][17][18].
Provozní zařízení mohou být ovládána lokálně na každém zařízení nebo centrálním inženýrským nástrojem. Všechny parametry zařízení lze uložit, zdokumentovat nebo upravit. To zlepšuje transparentnost a bezpečnost daného závodu[16][17][18].

Profibus FMS

Profibus FMS je univerzální řešení pro komunikační úkoly na buněčné úrovni. Výkonný FMS může spravovat širokou škálu aplikací a poskytovat velkou flexibilitu. V průmyslové oblasti dokáže FMS umožnit PLC zařízením komunikaci mezi sebou

a s inteligentními zařízeními. Pro FMS je nejdůležitější funkcionalita, než krátká reakční doba systému. Ve většině případech probíhá výměna dat cyklicky a na žádost uživatelského procesu[16][17][18].

1.3.3 Porovnání protokolů Modbus a Profibus

Oba protokoly mají své výhody či nevýhody. Například Profibus je schopen fungovat pouze na RS 485, na rozdíl od Modbusu, který dokáže pracovat na třech různých typech hardwarových vrstev.
Další vlastnost, která odlišuje Profibus protokol od Modbus, je režim více master jednotek. Ve srovnání s Modbus protokolem, který dokáže použít pouze jednu master jednotku, Profibus jich dokáže využívat více, což je možné díky zabudovanému dodatečnému protokolu, o kterém byla řeč výše. Další významné rozdíly těchto dvou protokolů jsou obsaženy v tab. 1.4[19].

Tab. 1.4: Porovnání Modbus a Profibus. Zdroj:[19]

Popis	Modbus	Profibus
Typ	Sériový komunikační protokol	Fieldbus standard pro komplexní automatizační aplikace.
Primárně používán	Navrženo pro průmyslové aplikace, pro přenos dat mezi řídícími zařízeními a senzory	Používá se především v automatizaci pro připojení senzorů, akčních členů a ovladačů v nebezpečných podmínkách.
Přenos dat	Master-slave protokol, kde hlavní jednotka iniciuje transakce (dotazy)	Funguje na principu předávání tokenu, kde hlavní zařízení předává token pro řízení komunikace.
Režimy	ASCII, RTU a TCP/IP	DP, PA a FMS
Rychlost	Až 19,2 Kbps v sériovém provedení, v TCP/IP i vyšší.	Až 12 MBps (Profibus DP)
Topologie	Jednoduchá sběrnicová nebo hvězdicová u Modbus TCP/IP	Profibus podporuje kruhovou, hvězdicovou a sběrnicovou topologii
Maximální počet zařízení	247 sériových zařízení, u TCP/IP režimu prakticky neomezeně	Až 126 zařízení
Kontrola chyb	CRC v RTU režimu, LRC v ASCII režimu	CRC
Aplikační vrstva	Jednoduché operace čtení/zápis	Komplexnější, vhodné pro širokou škálu aplikací
Interoperabilita	Vysoká díky své jednoduchosti a širokému použití	Vysoká, ale ne tak komplexní. Je více zaměřen na konkrétní produkty
Cena	Obecně menší	Obecně vyšší

2 Mobilní aplikace

V této moderní době informačních a komunikačních systémů je již standardem používat mobilní aplikace. I přes to, že jsou mobilní aplikace stále relativně nová věc, tak jejich vývoj a použití velmi rychle roste. Mají velmi pozitivní, globální dopad. Mobilní aplikace běží na malém mobilním zařízení, které je přenosné, snadno použitelné a dostupné odkudkoliv, což jsou jejich největší přednosti.
V dnešní době již většina lidí využívá mobilní aplikace ke kontaktování přátel, procházení internetu, správě obsahu souborů, vytváření mediálního obsahu, zábavě, či fotografování. Nejvíce se využívají v mediální nebo herní sféře, podnikání, nebo gastronomii. Naopak zatím největším nedostatkem mobilních aplikací, je jejich velmi malé využití v průmyslu.

Typy mobilních aplikací

iOS, Android a také Windows jsou jen některé z dostupných digitálních zařízení. Mobilní aplikace mají různé tvary, velikosti a funkce. Z této funkční oblasti lze tedy mobilní aplikace rozdělit do tří kategorií:

- **Nativní aplikace** jsou určeny pro konkrétní operační systém, jako je iOS, Android nebo Windows. Aplikace využívají funkce zařízení, jako je RAM, fotoaparát, nebo GPS[20].
- **Webová aplikace** je software, který je uložen na vzdáleném místě a distribuován přes web pomocí rozhraní prohlížeče, jako je Chrome, Mozila, či Edge. Klasickými příklady jsou například e-maily, online nákupy, aukční weby, nebo aplikace pro zasílání zpráv[20].
- **Hybridní aplikace** jsou aplikace, které se spouštějí na chytrém telefonu stejným způsobem jako všechny ostatní aplikace. To, čím jsou jedinečné, je, že kombinují funkce z nativních aplikací s komponenty z webových aplikací. K vytváření těchto aplikací se běžně používají tzv. „frameworky“[20].

2.1 Uživatelské rozhraní

Uživatelské rozhraní (UI) je tzv. styčný bod mezi lidmi a počítači. Každá technologie, se kterou uživatel komunikuje, je součástí tohoto UI[21].
Například obrazovky, zvuky, celkový styl, nebo odezva jsou prvky uživatelského rozhraní. UI obsahuje následující čtyři součísti:

- **Navigační prvky** pomáhají uživatelům orientovat se v rozhraní. Příklady navigačních prvků zahrnují různé posuvníky, šipky zpět, či vyhledávací pole[21].

- **Vstupní ovládací prvky** jsou prvky, které umožňují uživatelům zadávat informace. Jsou to různá tlačítka, zaškrtávací políčka nebo textová pole[21].
- **Informační složky** slouží ke sdělování informací uživateli. Patří sem například ukazatel průběhu nebo dialogové okno[21].
- **Kontejnery**, které organizují obsah do snadno stravitelných sekcí[21].

UI/UX

Při průzkumu tvorby UI designu se lze setkat se souvisejícím pojmem „uživatelské zkušenosti“ (UX). I když UI a UX sdílejí uršité podobnosti, existuje několik důležitých rozdílů viz.tab. 2.1[21].

Tab. 2.1: UI a UX. Zdroj:[21]

UI	UX
Zaměřuje se na design interakce, vizuální prvky webové stránky, nebo mobilní aplikace, zajišťuje že je navigační cesta vizuálně atraktivní a snadno použitelná	Zaměřuje se na uspokojení záměru uživatele a poskytuje jasnou navigační cestu pro přístup k informacím na webu, nebo v aplikaci

Typy UI

Návrh UI je pravděpodobně první věcí, se kterou se setkáte při interakci s mobilní aplikací nebo webovou stránkou. Design UI je zodpovědný za vzhled produktu, interaktivitu, použitelnost, chování a celkový dojem. Návrh uživatelského rozhraní může určit, zda má uživatel pozitivní zkušenost s produktem, takže je nezbytné, aby se společnost a tvůrci seznámili s osvědčenými postupy návrhu UI. Existují různé typy UI viz.tab. 2.2[21].

Tab. 2.2: Typy UI. Zdroj:[21]

Typy UI	Definice
Grafické uživatelské rozhraní (GUI)	Grafické uživatelské rozhraní umožňuje uživatelům komunikovat se zařízením prostřednictvím grafických ikon. Interakce jsou obvykle usnadněny pomocí myši, trackpadu nebo jiného nástroje typu ukázat-a-kliknou(point-and-click).
Hlasové uživatelské rozhraní (VUI)	Slova a syntaxe hrají nejdůležitější roli v hlasových uživatelských rozhraních. VUI používá rozpoznávání řeči k porozumění hlasovým příkazům. Mezi pozoruhodné příklady VUI patří Siri iPhone, funkce „hey google“ Google Home a Alexa od Amazonu.
Rozhraní řízené pomocí menu	Rozhraní řízená nabídkami poskytují uživatelům možnosti příkazů prostřednictvím seznamu nebo nabídky. Tyto příkazy se mohou zobrazovat na celé obrazovce, nebo jako vyskakovací nebo rozevírací nabídka.

Nástroje pro UI/UX design

Pro návrh UI je nezbytné mít ty správné nástroje a technologie. V tab. 2.3 jsou nastíněné 3 nástroje pro návrh UI/UX a uvedena cena a funkce[21].

Tab. 2.3: Nástroje UI/UX. Zdroj:[21]

UI nástroje	Cena	Vlastnosti
Figma	Částečně zdarma	Pokročilé nástroje pro kreslení, automatické rozvržení, styly, pluginy a widgety, import skic, interaktivní prototypy, jednoduché a uživatelsky přívětivé.
Penpot	Zdarma	Plně open-source, což znamená, že je možné, aby ho sama komunita dále vylepšovala. Obsahuje taktéž pokročilé nástroje pro kreslení, avšak méně, komunitou vytvořené, pluginy pro zvorku UI a UX.
Sketch	Placený	Vestavěná kontrola pravopisu, podpora barev, symboly, styly, barevné proměnné, testování prototypu prohlížeče, pluginy, exporty ve více měřítcích

Porovnání UI nástrojů

Jelikož se zaměřuji na neplacené nástroje, mezi které Sketch nespadá, je tedy na místě spíše srovnat Figmu a Penpot. Penpot je skvělý nástroj pro vývoj a návrh UI/UX, hlavně z toho důvodu, že je plně open-source a plně zdarma. Narozdíl od Figmy podporuje vytváření více projektů zároveň a podporuje neomezené využívání všech UI komponent[21][22].
Na druhou stranu, pokud je potřeba výkonnějšího nástroje, tak je určitě favoritem Figma. Díky funkcím prototypování, spolupráce v reálném čase, sdílení souborů je skvělou volbou. Díky delší funkci na trhu má již rozšířenější komunitu a tudíž podporuje větší množství různých UI balíčků a pluginů. Naopak Penpot komunita se velmi rychle rozvíjí. Přidávání nových balíčků a pluginů je tedy pouze otázkou času. Avšak některé pokročilejší funkce, jako je vytváření více projektů zároveň, nebo neomezené využívání komponent, již Figma nepodporuje ve verzi zdarma, tudíž není pro jednodušší a osobní projekty tolik efektivní, jako Penpot. V tab. 2.4 jsou vyzdvižena některá další srovnání nástrojů Figma a Penpot[21][22].

Tab. 2.4: Figma vs Penpot. Zdroj:[22]

	Figma	**Penpot**
Kompatibilita nástrojů	Photoshop, Slack, Trello, Sketch,...	-
Podporované platformy	Web, Windows, macOS, Linux, iOS, Android	Web, Windows, macOS, Linux
Rychlost načítání, vykreslování	Velmi rychlé	Rychlé
Neomezené uložiště	NE	ANO
Neomezené využití UI komponent	NE	ANO

2.2 Framework

Je k dispozici řada nástrojů pro vývoj mobilních aplikací. Společnosti, které mají, nebo chtějí vytvořit novou mobilní aplikaci, potřebují hloubkovou studii, strategii, komplexní plánování a dobrou analýzu současného mobilního prostředí, aby mohly na mimořádně agresivním digitálním trhu vzkvétat[23].
Nativní aplikace jsou taktéž drahé na zavedení a údržbu. U nativních aplikací je největší problém kompatibilita z důvodu velké rozmanitosti mobilních zařízení. Odpovědí na tento problém je použití různých nástrojů pro vývoj a údržbu mobilních aplikací, jako jsou například frameworky[23].

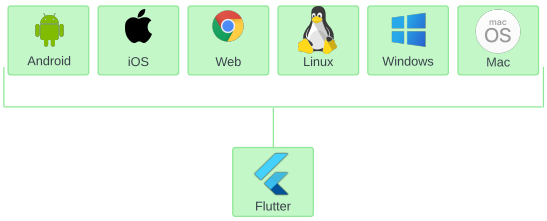
Definice

Rámec pro mobilní aplikace (framework) je platforma pro vytváření softwaru, která mimo jiné zahrnuje nástroje a software, kompilátory, ladící nástroje a programovací rozhraní. Vývojář tedy vytváří zdrojový kód aplikace a framework používá různé

prvky k generování aplikace pro různá mobilní zařízení. V každém případě záleží na primárních cílech a finančních omezeních dané firmy nebo jedince. Samotné nativní aplikace jsou z hlediska ceny nejdražší. Pro méně finančně náročné řešení jsou hybridní aplikace. Dalším významným parametrem je okruh uživatelů, pro které je aplikace určena. Pro větší okruh uživatelů jsou taktéž lepší hybridní aplikace. Naopak v rámci menšího okruhu uživatelů je nejlepší volba nativní mobilní aplikace, z důvodu rychlosti a lepší udržitelnosti[23].

2.2.1 Flutter

Flutter je framework s otevřeným zdrojovým kódem (open-source) pro vytváření vizuálně atraktivních, nativně kompilovaných, multiplatformních aplikací z jediné kódové základny.
Flutter kód se kompiluje do strojového kódu ARM nebo Intel, stejně jako například JavaScript, velký výkon na jakékoliv zařízení[24].
Podporuje tzv. „Hot Reload“, což znamená, že lze téměř okamžitě, bez ztráty stavu, vidět změny, které byly v kódu provedeny[24].
Díky tomuto frameworku lze ovládat každý pixel a vytvářet tak adaptivní návrhy, které jsou kompatibilní s jakoukoliv obrazovkou[24].
Není avšak využitelný, jen co se týče grafického a uživatelského rozhraní. Díky své obsáhlé komunitě a skutečnosti, že je open-source, je čím dál vhodnější i pro průmyslové aplikace, jako je například komunikace a řízení robotických rukou pomocí průmyslového Modbus protokolu[24].
Jedním z hlavních cílů společnosti Flutter je vytvořit framework, který umožní vyvíjení aplikací z jedné kódové základny, které lze spustit na jakékoliv platformě viz obr. 2.1[24].



Obr. 2.1: Multiplatformní framework. Zdroj:[24]

UI

Ústřední myšlenkou je, že se ve Flutteru tvoří UI pomocí tzv. „widgetů“. Widgety popisují, jak by měl jejich pohled vypadat vzhledem k jejich aktuální konfiguraci

a stavu. Když se změní stav widgetu, tak tento widget sestaví znovu svou novou podobu, kterou Flutter porovná s předchozím stavem, aby určil minimální změny potřebné k vykreslení přechodu z jednoho stavu do dalšího[24].
Mezi základní widgety patří:

- **`Text`** – umožňuje vytvářet řadu stylizovaných textů[24].
- **`Row`**, **`Column`** – jsou flexibilní widgety, které umožňují spojení více widgetů v horizontálním (Row) nebo vertikálním (Column) směru[24].
- **`Stack`** – umožňuje, na rozdíl od lineární orientace, umístit widgety na sebe, tedy jejich zakrývání. V rámci tohoto widgetu lze využít widget `Positioned`, pro přesnější definici polohy (horní část, dolní, nebo pravý roh[24].
- **`Container`** – umožňuje vytvořit obdélníkový vizuální prvek. Tento prvek lze jakkoliv vizuálně poupravit (barva pozadí, okraje, stíny) například jedním z jeho parametrů `BoxDecoration`. Dále mu lze přiřadit různé funkce a interakce po kliknutí nebo podržení[24].

Rozložení UI

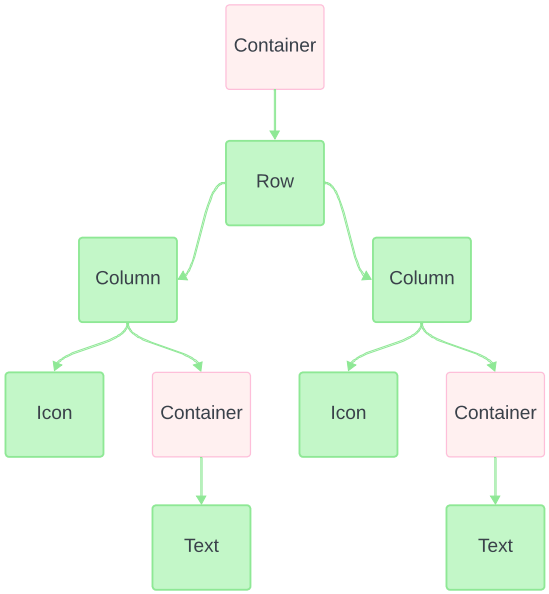
Rozložení lze vytvořit složením více widgetů do jednoho celku. Na obr. 2.2 lze vidět 2 ikony a každá má pod sebou svůj štítek[24].

Obr. 2.2: Celek widgetů. Zdroj:[24]

Obrázek 2.3 zobrazuje vizuální blokový diagram rozvržení těchto ikon v rámci widgetů, kde každý sloupec obsahuje ikonu a štítek. Reálné kódové řešení tohoto příkladu lze vidět na výpisu 2.1

Výpis 2.1: Celek widgetů. Zdroj:[24]

```
Container(
    child: Row(
          children: [
            Column(
              children: [
                Icon(Icons.phone),
                Container(
                  child: Text("Hovor"),
                )],
            ),
            Column(
              children: [
                Icon(Icons.share),
                Container(
                  child: Text("Sdílení"),)],)],),)
```



Obr. 2.3: Blokový diagram widget celku. Zdroj:[24]

UX

UX je ve Flutteru řešeno pomocí navigačních widgetů. Flutter poskytuje kompletní systém pro navigaci mezi obrazovkami a taktéž systém pro přímé odkazování. U malých aplikací, bez složitých přímých odkazů, lze využít **`Navigator`** widget. Zatímco u komplexnějších aplikací se specifickými požadavky na přímé odkazy a navigaci se využívá **`Router`** widget. Tento widget se využívá například ve vývoji webových aplikací, pro zajištění synchronizace s adresním řádkem [24].

- **`Navigator`** – widget zobrazuje obrazovky jako zásobních pomocí přechodových animací pro cílovou platformu jako je znázorněno ve výpisu 2.2. Jelikož

`Navigator` uchovává zásobník objektů Route(představující zásobních historie), tak metoda `push()` také přebírá objekt `Route`. Objekt `MaterialPageRoute` je podtřídou `Route`, která určuje přechodové animace z `Material Design` balíčku[24].

- `Route` řešení je další řešení, pokud je potřeba interakce s adresním řádkem. Implementace je zobrazena na výpisu 2.3[24].

Výpis 2.2: Použití `Navigator` widgetu. Zdroj:[24]

```
    onPressed: () {
  Navigator.of(context).push(
    MaterialPageRoute(
      builder: (context) => const SecondScreen(),),);},
child: Text("Název"),
```

Výpis 2.3: Použití `Route` widgetu

```
    onPressed: () {
  Navigator.of(context).push(
    MaterialPageRoute(
      builder: (context) => const SecondScreen(),),);},
child: Text("Název"),
```

Animace a přechody

Další součástí UX jsou animace a přechody. Dobře navržené animace činí aplikaci intuitivnější, přispívají k hladkému vzhledu a zlepšují uživatelský dojem. Flutter podporuje širokou škálu animací, což usnadňuje implementaci. Mnoho widgetů, jako je například `Material`, přichází se standardními pohybovými efekty definovanými v jejich specifikaci, ale je ovšem možné tyto animační efekty přizpůsobit [24]. Mezi takové widgety patří například:

- `AnimatedDefaultTextStyle`
- `TweenAnimationBuilder`
- `AnimatedBuilder`
- `CustomPainter`

Serverová část

Ve Flutteru lze komunikovat se vzdáleným serverem pomocí HTTP protokolu, konkrétně pomocí HTTP požadavků [24].
Tyto HTTP požadavky lze z Flutter aplikace generovat pomocí balíčku (`package`) `http`[24].
Pro komunikaci se serverem a následné manipulaci s daty se využívají různé metody:

- **`get`** – pro načtení dat ze serveru
- **`post`** – pro odesílání dat na server
- **`put`** – pro aktualizaci dat na serveru

- `delete` – pro odstranění dat ze serveru

Serializace

Serializace, neboli **dekódování dat**, se ve Flutteru řeší metodou JSON. Jednoduše se data ze serveru převedou do JSON formátu, který je již pro Flutter zpracovatelný[24]. Jsou dva typy JSON serializace:

- **Manuální serializace**
- **Automatizovaná (generovaná) serializace**

Různé typy projektů mají různé složitosti a různé případy použití. U menších a méně komplexních aplikací by mohlo být zbytečně přehnané použití generované serializace. Naopak v komplexnějších a větších aplikacích může být ruční JSON serializace únavná a opakující se a může při ní docházet i k malým chybám. **Manuální serializace** se týká použití vestavěného JSON dekodéru v balíčku `dart:convert`. Zahrnuje předání, nezpracovaného JSON řetězce, funkci `jsonDecode()` a následné vyhledání hodnot, které jsou potřeba ve výsledné mapě dynamických řetězců. Nemá žádné externí závislosti ani konkrétní nastavení procesu a je dobrý pro rychlé ověření konceptu. **Automatizovaná serializace** znamená, že externí knihovna vygeneruje základní kód sama. Po počátečním nastavení se spustí nástroj pro sledování souborů, který generuje kód z vytvořených modelových tříd v rámci aplikace. Mezi používané knihovny patří například `json_serializable` nebo `built_value`. Tento přístup je vhodný u komplexnějších aplikací. Není zde třeba žádné ruční dekódování a případné překlepy při přístupu k JSON polím jsou zachyceny během kompilace. Možnou nevýhodou této automatizované serializace je, že vyžaduje určité počáteční nastavení. Příklad kódu pro načtení dat ze serveru a následné serializace, či deserializace, lze vidět na výpisu 2.4[24].

Výpis 2.4: Metoda `get` a serializace. Zdroj:[24]

```
1    Future<Album> fetchAlbum() async {
2  final response = await http
3      .get(Uri.parse(
4        'https://jsonplaceholder.typicode.com/složka/1'));
5  if (response.statusCode == 200) {
6    // Pokud server vrátí status 200 OK
7    // tak se provede dekódování těla dat
8    return Album.fromJson(jsonDecode(response.body)
9        as Map<String, dynamic>);
10 } else {
11   // Pokud server nevrátí status 200 OK,
12   // tak vrátí vyjímku
13   throw Exception('Nepodařilo se načíst data');}}
```

Externí balíčky a doplňkové zásuvné moduly

Flutter podporuje používání sdílených balíčků a různých doplňkových zásuvných modulů, přidaných jinými vývojáři do Flutter ekosystému. To taktéž napomáhá k rychlému vývoji různých aplikací a všestrannému využití frameworku Flutter. **Balíčky** mohou obsahovat různé aplikace, prostředky, testy, obrázky, animace či fonty. **Doplňkové zásuvné moduly** jsou speciálním druhem balíčku, který aplikaci zpřístupňuje funkce platformy. Mohou být psány pro Android, iOS, web, macOS, Windows, Linux a nebo pro jakoukoliv jejich kombinaci. Mohou například poskytnout Flutter aplikacím využívat fotoaparát zabudovaný v zařízení [24].

Testování a ladění

Čím více funkcí a objektů má aplikace, tím těžší je ruční testování. Automatizované testy pomáhají zajistit, aby aplikace fungovala správně, než bude publikována, a přitom zachovají veškerou funkcionalitu [24].
Automatizované testování spadá do několika kategorií:

- **jednotkové testy**, které se používají k testování jedné funkce, metody nebo třídy
- **widget testy** (test komponent) testují jeden widget
- **integrační testy**, které testují kompletní aplikaci nebo velkou část aplikace

Obecně řečeno, dobře otestovaná aplikace potřebuje mnoho jednotkových testů a widget testů, pro sledování a kontrolu menších částí kódu a dostatek integračních testů pro pokrytí většího množství, na sobě závislých, částí kódu. Tato skutečnost naznačuje určité vlastnosti a různé situace využití těchto testů, jak je vidět v tabulce 2.5[24].

Tab. 2.5: Vlastnosti testů. Zdroj:[24]

Vlastnosti	Jednotkový test	Widget test	Integrační test
Spolehlivost	Nízká	Vyšší	Nejvyšší
Náklady na údržbu	Nízká	Vyšší	Nejvyšší
Závislosti	Pár	Více	Nejvíce
Rychlost	Nejvyšší	Vyšší	Nízká

2.2.2 React Native

React Native (React) je JavaScriptový framework pro vytváření UI. Využívá se k vytváření jednostránkových aplikací a umožňuje vytvářet opakovaně použitelné UI komponenty[25][26].

Vytvoření komponent

Jako bylo vše ve Flutteru vytvořeno pomocí widgetů, tak v Reactu se vše skládá z komponent. Komponenta je část UI, která má vlastní logiku a vzhled. Může být malá, například v podobě tlačítka, nebo větší v podobě celé stránky. Komponenty Reactu vycházejí z funkcí JavaScriptu, jak je ukázáno níže na výpisu 2.5[25][26].

Výpis 2.5: Malá komponenta React. Zdroj:[26]

```
1    function MojeTlačítko() {
2   return (
3     <button>Tlačítko Reactu</button>);}
```

Syntaxe značek, kterou je možno vidět ve výpisu 2.5 se nazývá JSX. V React je možné používat i jiné značkové syntaxe, ale jelikož je JSX podporován frameworkem React nativně, tak je to ta nejpohodlnější volba. JSX se velmi podobá HTML[25][26].

Stylizace

V Reactu se stylizace řeší podobně jako u CSS. Třída CSS se zadává pomocí className a funguje podobně, jako atribut HTML, což lze vidět na výpisu 2.6[25][26].

Výpis 2.6: Atribut React. Zdroj:[26]

```
1    <img className="avatar" />
```

a poté se pro tento atribut zapíší pravidla CSS do samostatného souboru CSS viz výpis 2.7[25][26].

Výpis 2.7: CSS stylizace React. Zdroj:[26]

```
1    .avatar {
2   border-radius: 50%;
3 }
```

Zobrazení dat

JSX umožňuje vkládat značky do JavaScriptu. Složené závorky umožňují „utéct zpět“ do JavaScriptu, aby bylo možné vložit nějakou proměnnou z kódu a zobrazit ji uživateli. Příklad pro zobrazení `admin.name` viz výpis 2.8[25][26].

Výpis 2.8: Zobrazení dat. Zdroj:[26]

```
1    return (
2   <h1>
3     {admin.name}
4   </h1>
5 );
```

Podmíněné vykreslování

React nemá žádnou speciální syntaxi pro podmíněné vykreslování. Namísto toho se používají stejné techniky, jako u JavaScriptu. Příklad použití podmínky `if` viz výpis 2.9 [25][26].

Výpis 2.9: Podmíněné vykreslování. Zdroj:[26]

```
   let something;
if (isLoggedIn) {
  something = <AdminKarel />;
} else {
  something = <LoginForm />;
}
return (
  <div>
    {something}
  </div>
);
```

Lze taktéž využít podmíněného operátoru `?` a `:`.

2.2.3 Porovnání Flutter a React

Obecně

Flutter a React jsou známé jako frameworky té nejvyšší úrovně pro vývoj mobilních a webových aplikací napříč platformami. Flutter se může chlubit robustními možnostmi uživatelského rozhraní a pěkným designem zaměřeným na uživatele, zatímco React má velkou výhodu z hlediska vyspělosti a rozsáhlé komunity. Oba jsou to skvělé frameworky, ale u jejich výběru velmi záleží na specifických potřebách. Jejich specifické rozdíly jsou zobrazeny v tab. 2.6 [27].

Tab. 2.6: Porovnání Flutter a React Native. Zdroj:[27]

	Flutter	**React Native**
Programovací jazyk	Dart	JavaScript
Význam pro vývojáře	Urychluje vývoj aplikací a snižuje náklady a potíže s vytvářením aplikací napříč platformami. Má rychle rostoucí komunitu vývojářů a silnou podporu od společnosti Google.	Základní logika je založena na knihovně Web React, které je ze všech nejpopulárnější. Vývoj aplikací je zde taktéž rychlý, avšak zde je potřeba psát zvlášť kód pro všechny platformy.
Nativní podpora uživatelského rozhraní	Je k dispozici, ale vyžaduje další programování, aby dokázalo poskytnout rozhraní specifické pro platformu.	UI je zde optimalizováno pro každou platformu zvlášť.
Konzistence uživatelského rozhraní	Předem připravené. UI je podobné na všech platformách a verzích operačního systému (OS).	Vyžaduje další práci/nástroje, aby bylo UI konzistentní. Aplikace mohou vypadat jinak v různých verzích OS
Výkon	Vyšší než v React Native díky přímé komunikaci s platformou.	Nižší než ve Flutteru kvůli tzv. mostu mezi nativní a Reactovou částí aplikace.
Podporované platformy	Umožňuje nativně plnou podporu platformám Android, iOS, Windows, Linux, MacOS, Web.	Nativně podporuje Android a iOS. Podporuje také Web a Windows, ale je potřeba externích nástrojů.
Dokumentace	Snadno čitelná, skvěle organizovaná, i přes to, že je velmi obsáhlá.	Méně čitelná a organizovaná než u Flutteru. Avšak díky své rozsáhlé komunitě lze mnoho odpovědí najít na fórech pro vývojáře.

Využití v průmyslových sítích

Tyto frameworky lze využít taktéž pro průmyslové sítě. Konkrétně například pro generování průmyslového protokolu Modbus a všech jeho typů. Oba frameworky bohužel tuto možnost nepodporují nativně, ale pomocí externích balíčků. Avšak zde vzniká hlavní rozdíl mezi frameworkem Flutter a React Native. React Native totiž není schopen zpracovávat Modbus přímo v aplikaci. Je nutné vytvořit externí backendový kód, který bude s React aplikací komunikovat[28]. Na rozdíl od toho Flutter je schopen Modbus generovat i zpracovávat přímo v aplikaci. Pro Flutter, respektive Dart, existují balíčky jako:

- `modbus_client_tcp`
- `modbus_client`
- `modbus`

2.3 Integrované vývojové prostředí

Integrované vývojové prostředí (IDE) je software pro programování aplikací, který kombinuje běžné vývojářské nástroje do jediného grafického uživatelského rozhraní (GUI). Obvykle se skládá z:

- **Editor zdrojového kódu** – textový editor, který může pomoci při psaní softwarového kódu pomocí funkcí, jako je zvýraznění syntaxe pomocí vizuálních podnětů, poskytování možnosti automatického dokončování specificky pro daný, nebo kontrola chyb při psaní kódu[24].
- **Automatizace lokálního sestavování** – nástroje které automatizují jednoduché, opakované úkoly pro použití vývojářem, jako je například kompilace zdrojového kódu na binární, či spouštění testů[24].
- **Ladící program** – je program, pro testování jiných programů, který dokáže graficky zobrazit umístění chyby v původním kódu [24].
- **Emulátor** – je virtuální zařízení (mobilní telefon, tablet), které lze používat pro testování a kontrolu vizuálního stavu aplikace. Konkrétně Android Studio jej podporuje nativně viz obrázek 2.4 [24].

2.4 Nástroje pro správu zdrojového kódu

2.4.1 Git

Git je v IT specifikován jako distribuovaný systém správy verzí [29][30].
Využívá se pro:

Obr. 2.4: Emulátor Android Studio

- **Sledování změn kódu** – pomocí funkcí `Staging` a `Committing` dokáže kontrolovat čas změn kódu, konkrétní místo nebo autora[29][30].
- **Spolupráce na tvorbě kódu** – pomocí možnosti „větvení“ je možné, aby na jednom zdrojovém kódu pracovalo více vývojářů. Pomocí metody `PULL` je možné si lokálně zkopírovat nejnovější verzi zdrojového kódu. Vytvořit na této kopii změny, které se po dokončení zkontrolují a vyhodnotí, které z nich budou použity a následně spojeny s hlavním kódem pomocí metody `PUSH`[29][30].
- **Spravování projektů pomocí repozitářů** – obecně uložiště. Slouží pro zálohu souborů na vzdálených serverech. V praxi může být nebezpečné ukládat si soubory pouze lokálně a proto existují takovéto služby[29][30].

2.4.2 GitHub

GitHub je cloudová platforma, kde lze ukládat, sdílet a spolupracovat s ostatními vývojáři na vývoji zdrojového kódu[29][30].
Uložení kódu do GitHub uložiště umožňuje:

- Sdílení zdrojového kódu.
- Sledování a spravování změn kódu v průběhu času.
- Možnost kontroly kódu a jeho následného vylepšování.
- Nezávislá spolupráce na stejném projektu v práci více programátorů.

Tyto všechny možnosti jsou umožněny díky softwaru Git, na kterém se GitHub postaven[29][30].

2.4.3 Rozdíl mezi Git a GitHub

Hlavním rozdílem mezi nimi je to, že Git je systém, který umožňuje spravovat a sledovat historii zdrojového kódu. Avšak jeho hlavní nevýhodou je to, že se s ním pracuje prostřednictvím příkazové řádky, což může být v určitých situacích zdlouhavé a náročné. Z toho důvodu vznikl GitHub, díky kterému lze Git spravovat prostřednictvím grafického rozhraní. Z důvodu, že většina vývojářů pracuje na svém kódu lokálně (na svém počítači), tak je nutné všechny změny – a všechna data související s Gitem – neustále synchronizovat se „vzdáleným“ uložištěm na GitHub. Pro toto využití existuje spousta nástrojů, mezi které patří například GitHub Desktop[29][30].

2.5 Flask

Flask je malý a lehký webový Python framework, který poskytuje užitečné nástroje a funkce, které usnadňují vytváření serverů (backendu) pro mobilní a webové aplikace. Poskytuje flexibilitu a je velmi přístupným frameworkem pro nové vývojáře, jelikož dokáže pouze jediným souborem vytvořit funkční server. Je taktéž dobře rozšířitelný a nevynucuje si konkrétní adresářovou strukturu ani nevyžaduje komplikovaný kód[31].

Příklad aplikace

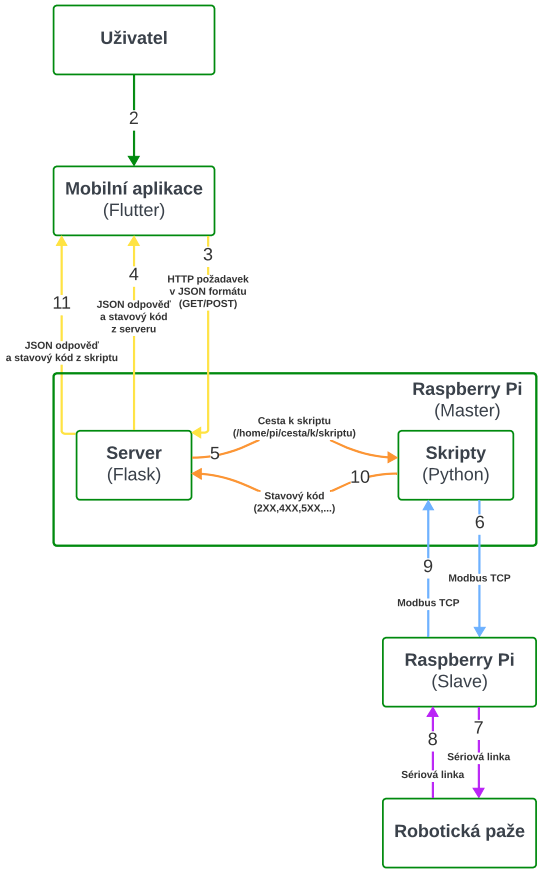
Pro vytvoření a spuštění serveru stačí ve Flask frameworku pár řádků kódu, jak je vidět na výpisu 2.10.
Nejdříve se vloží potřebné balíčky. Nejdůležitější je balíček `Flask`, díky kterému flask server funguje. Dále `jsonify`, který převádí Python data do JSON formátu (což se lépe zpracovává v mobilních nebo webových aplikacích z důvodu HTTP komunikace). A nakonec `request` ke snadnému provádění HTTP požadavků a odpovědí (GET, POST, DELETE, ...). Dále je vytvořena instance této třídy. První argument je název modulu nebo balíčku aplikace, tedy `__name__`. Je to zkratka, která se používá ve většině případech, aby Flask věděl, kde hledat zdroje jako šablony a statické soubory[31].
Dále je použit tzv. „endpoint“ (koncový bod) `route()` k tomu, aby Flasku bylo řečeno, jaká URL má danou funkci spustit. Následně funkce vrací zprávu, která se zobrazuje v prohlížeči uživatele. V tomto případě to budou data ze skriptu v JSON formátu [31].
Na konci je metoda pro spuštění Flask serveru, konkrétně na portu 5001 a je dostupný ze všech IP adres v síti[31].

Výpis 2.10: Příklad základního Flask serveru. Zdroj:[31]

```
from flask import Flask, jsonify, request

app = Flask(__name__)

@app.route('/test-master', methods=['GET'])
def test_master():
    return jsonify(run_script(
        "/home/pi/scripts/test-master.py"))

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5001)
```

3 Prvotní návrh realizace

Pro řešení bylo nejdříve potřeba navrhnout základní komunikaci mezi mobilní aplikací a robotickou paží.
Jelikož framework Flutter primárně podporuje komunikaci přes HTTP protokol, tak bylo potřeba vytvořit server, který by tyto HTTP požadavky zpracovával. Bylo tedy využito frameworku Flask pro vytvoření prostředí pro zprávu serveru a komunikaci s mobilní aplikací.
Dále bylo potřeba vyřešit komunikaci mezi serverem a robotickou paží. Zde bylo využito již vytvořených skriptů, které ovládají robotickou paži pomocí protokolu Modbus/TCP. Tedy server Flask je schopen spouštět skripty, které následně řídí robotickou paži.
Pro lepší odezvu je server i skript uloženy v mikropočítači Raspberry Pi. Blokové schéma realizované základní komunikace lze vidět na obr. 3.1.



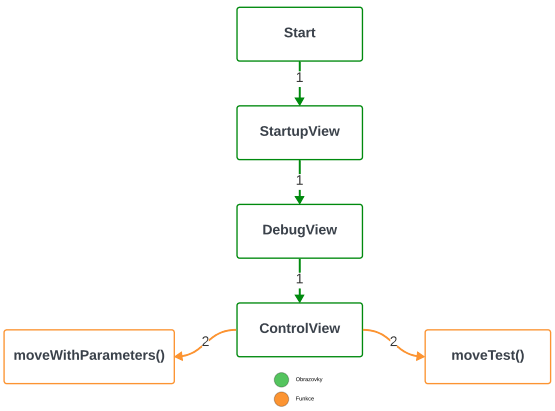
Obr. 3.1: Blokové schéma prvotního návrh realizace

3.1 Popis realizace základní komunikace

- **Uživatel** – položka uživatele interpretuje reálného uživatele mobilního zařízení, který interaguje s mobilní aplikací (**2**)
- **Mobilní aplikace** (Flutter) – naprogramované aplikace, která komunikuje s Flask serverem. Tato mobilní aplikace zpracuje požadavek od uživatele (kliknutí na příslušné tlačítko) a odešle příslušný HTTP požadavek GET nebo POST v JSON formátu na server (**3**).
- **Server** (Flask) – server, který zpracovává HTTP požadavky z mobilní aplikace. Následně vrátí odpověď mobilní aplikaci o stavu požadavku (zda proběhl úspěšně, či nikoliv) a informace o tom, jakou akci provede, taktéž v JSON formátu (**4**). Zároveň, na základě požadavku z mobilní aplikace, spustí příslušný skript (**5**).
- **Skripty** – již hotové skripty odesílají stavový kód zpět serveru na základě toho, zda se spustili správně a nebo ne a taktéž samotná data, která obsahují (**10**). Zároveň generují Modbus TCP, díky kterému komunikuje s dalším Raspberry Pi. (**6**).
- **Robotická paže** – realizuje výsledný reálný pohyb a činnost a zároveň odesílá stavová data zpět ke skriptu (**7**).

3.2 Vytvoření prvotní mobilní aplikace

Pro základní realizaci je dále potřeba naprogramovat mobilní aplikaci, která umožní uživateli interakci s robotickou paží, viz obr. 3.2



Obr. 3.2: Realizace prvotní mobilní aplikace Flutter

Popis realizace základní mobilní aplikace Flutter

- **Start** – reprezentuje prvotní interakci uživatele s mobilní aplikací. Kliknutím na ikonu mobilní aplikace, na ploše mobilu se aplikace spustí a pomocí metody `Navigator` (**1**) se spustí první obrazovka – StartupView.
- **StartupView** – první obrazovka, která „vítá“ uživatele, zobrazuje jméno aplikace a animaci, která reprezentuje načítání aplikace. Po načtení se znovu pomocí metody `Navigator` (**1**) se zamění za obrazovku – DebugView.
- **DebugView** – tato obrazovka slouží pro ulehčení UX při vývoji aplikace a navigaci mezi všemi obrazovkami. Obrazovky jsou zde reprezentovány pomocí `Container` tlačítek, díky kterým se pomocí `Navigátor` lze dostat na danou obrazovku.
- **ControlView** – je jedna z obrazovek, která konkrétně obsahuje metody, tlačítka a funkce, pro testování a vlastní ovládání robotické paže. Obsahuje funkce `moveTest()` viz výpis 3.1, která se spustí díky tlačítku `Test` viz výpis 3.2 (**2**) a slouží pro testování komunikace mezi aplikací a robotickou paží – spustí skript, ve kterém jsou předdefinované testovací parametry, které určují, jakým způsobem se robotická paže pohne.
  Dále obsahuje funkci `moveWithParameters()`, která slouží již pro vlastní ovládání robotické paže. Je speciálně upravena tak, aby bylo možné spouštět skript s vlastními parametry. Tato funkce je vytvořena pro duplikování v rámci celé aplikace, pouze s možností vlastního definování konkrétních pohybových parametrů robotické paže.

UI a UX

Základní návrh UI a UX byl vytvořen v nástroji Penpot viz obr. 3.3.

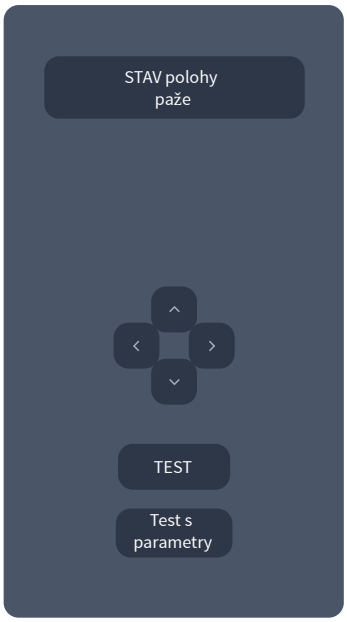
Programování

Dále je třeba dle UI, UX a realizace mobilní aplikace naprogramovat samotnou mobilní aplikaci.

Založení projektu

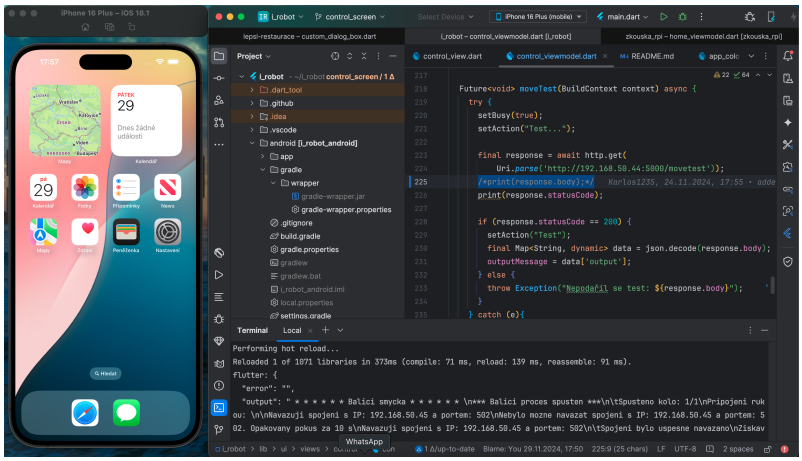
Pro založení projektu byl využit framework Flutter . Tento framework nabízí jednoduché vytvoření nového projektu pomocí příkazové řádky.

- Otevření konzole – terminálu – pomocí příkazu `cd` se vybralo místo, ve kterém se vytvoří nový projekt a následně se použil příkaz `stacked create app i_robot`. Tento příkaz dokáže vygenerovat nový projekt i se základní architekturou a základní, úvodní aplikaci.



Obr. 3.3: Základni návrh UI v nástroji Penpot

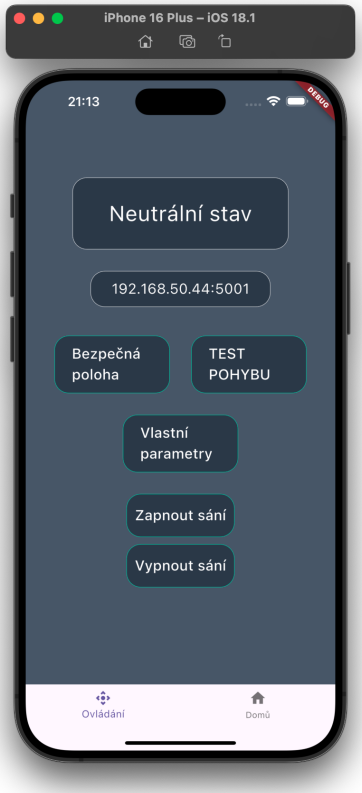
- Pro ukládání kódu aplikace a lepší srozumitelnost, byl projekt propojen s gitem a spravovan pomocí GitHubu. Pro jednoduchou implementaci gitu byl použit (ve složce, kde je uložen projekt) příkaz **git init** a v aplikaci Github Desktop vizuálně zobrazen kód této mobilní aplikace. Dále je pro všechny činnosti, v rámci gitu, využíván Github Desktop.
- Otevření projektu v IDE – dále byl projekt otevřen v Android Studiu – který byl vybrán z důvodu nativní podpory emulátoru viz obr. 3.4.



Obr. 3.4: Android Studio a emulátor

Vytvoření hlavní obrazovky

Pomocí příkazu `stacked create view control_view` byla vytvořena obrazovka control_view, která slouží pro testování a řízení robotické paže. Příkaz vygeneroval dva soubory – control_view a control_viewmodel. Oba soubory jsou navzájem propojené. V prvním souboru jsou naprogramované samotné widgety této obrazovky viz výpis 3.2. Ve druhém jsou naprogramované funkce a proměnné pro daná tlačítka a widgety viz výpis 3.1.
Tuto hlavní obrazovku lze vidět na obr. 3.5



Obr. 3.5: Hlavní obrazovka

Výpis 3.1: Funkce moveTest v mobilní aplikaci Flutter

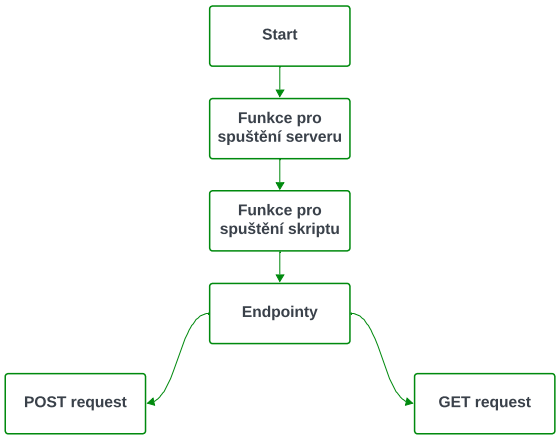
```
Future<void> moveTest(BuildContext context) async {
 try {
   setBusy(true);
   setAction("Test...");
   final response = await http.get(
       Uri.parse('http://192.168.50.44:5000/movetest'));
   /*print(response.body);*/
   print(response.statusCode);
   if (response.statusCode == 200) {
     setAction("Test");
     final Map<String, dynamic> data =
         json.decode(response.body);
     outputMessage = data['output'];
   } else {
     throw Exception("Nepodařil se test:
        ${response.body}");}
 } catch (e){
   setAction(neutralState);
   showDialog(
       context: context,
       builder: (ctx) => AlertDialog(
         title: Text("Nepodařil se test"),
         content: Text("$e"),
         actions: [
           MaterialButton(
             onPressed: () => Navigator.of(ctx).pop(),
             child: Text("OK"),
           )],));
 } finally {
   setBusy(false);
 }}
```

Výpis 3.2: Tlačítko Test v mobilní aplikaci Flutter

```
SizedBox(
     width: 160,
     height: 80,
     child: MaterialButton(
       onPressed: () => viewModel.moveTest(context),
       color: AppColor.primaryBackgroundColor,
       shape: RoundedRectangleBorder(
         borderRadius: BorderRadius.circular(20),
       ),
       child: Text(
           "TEST",
           style: TextStyle(
             fontSize: 20,
             color: AppColor.textColor,
           ),
         maxLines: 2,),),),
```

3.3 Vytvoření serveru pro zajištění komunikace mezi RaspberryPi a mobilní aplikací

Jelikož už je vytvořena základní část mobilní aplikace, tak je potřeba ještě navrhnout viz obr. 3.6 a naprogramovat server, který bude komunikovat s mobilní aplikací a se skripty.



Obr. 3.6: Realizace serveru Flask

Popis realizace serveru Flask

- **Start** – reprezentuje spuštění serveru.
- **Funkce pro spuštění serveru** – definuje port a IP, na které se daný server spustí vit výpis 3.3.
- **Funkce pro spuštění skriptu** – ve výpisu 3.4 je ukázán kód, který toto spuštění realizuje. Je v něm v definován příkaz, respektive tvar příkazu, který se musí zadat do příkazové řádky, aby byl skript spuštěn.
- **Endpointy** – jako /move-arm, nebo /move-arm-test, regují na HTTP požadavky (GET/POST) z mobilní aplikace viz výpis 3.5.

Výpis 3.3: Spuštění Flask serveru na RaspberryPi

```
1 from flask import Flask, request, jsonify
2 import subprocess
3 app = Flask(__name__)
4 # Spuštění serveru
5 if __name__ == '__main__': app.run(debug=True, host='0.0.0.0', port=5001)
```

Výpis 3.4: Spuštění Python skriptu na RaspberryPi

```
from flask import Flask, request, jsonify
import subprocess
app = Flask(__name__)
# Funkce pro spuštění skriptu
def run_script(script_path, args=None):
    try: command = ["python", script_path]
        if args: command.extend(args)
        result = subprocess.run(command, capture_output=True, text=True, check=True)
        return {"status": "success", "output": result.stdout}
    except subprocess.CalledProcessError as e:
        return {"status": "error", "error_message": str(e), "output": e.output}
```

Výpis 3.5: Endpointy Flask serveru

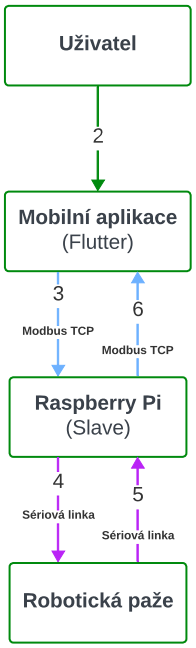
```
from flask import Flask, request, jsonify
import subprocess
# Endpoint pro GET požadavek (testování komunikace s paží)
@app.route('/move-test', methods=['GET'])
def test_arm():
    script_path = "/home/pi/ruce-showroom/mobilni_aplikace/slave_ridici_skript/
        test.py"  # Skript pro testování
    result = run_script(script_path)
    return jsonify(result)
# Endpoint pro POST požadavek (nastavení parametrů a spuštění paže)
@app.route('/move-arm', methods=['POST'])
def move_arm():
    data = request.get_json()
    if not data:
        return jsonify({"error": "No data provided"}), 400
    script_path = "/home/pi/ruce-showroom/mobilni_aplikace/slave_ridici_skript/
        aplikace_master.py"  # Skript pro nastavení parametrů
    args = [
        "-x", str(data.get("x", 0)),
        "-y", str(data.get("y", 0)),
        "-z", str(data.get("z", 0)),
        "-r", str(data.get("r", 0)),
        "-l", str(data.get("l", 0))
    ]
    result = run_script(script_path, args)
    return jsonify(result)
```

4 Realizace finální komunikace

Z důvodu zpoždění při komunikaci a náročnosti správy a vývoje byl navržen jiný způsob realizace komunikace mezi mobilní aplikací a robotickou paží.
Díky externímu balíčku, který framework Flutter nabízí, je možné generovat protokol Modbus/TCP přímo z mobilní aplikace. Aplikace tedy komunikuje přímo s Modbus serverem, který je na Raspberry Pi, a zapisuje hodnoty do příslušných registrů, což umožňuje pohyb robotické paže viz tab. 4.1. Nyní již není potřeba žádného Flask serveru ani skriptů. Blokové schéma finální komunikace lze vidět na obr. 4.1.

Tab. 4.1: Organizace paměti robotické paže

		Coil	Holding register
Současná pozice	pozice-x		1
	pozice-y		2
	pozice-z		3
	pozice-r		4
	pozice-l		5
Cílová pozice	pozice-x		6
	pozice-y		7
	pozice-z		8
	pozice-r		9
	pozice-l		10
Stav	klid/pohyb (0=klid)	1	
Stav sání	klid/sání (0=klid)	2	
Stav dopravníku	klid/pohyb (0=klid)	3	
Stav kolejnice	klid/pohyb (0=klid)	5	
Barva RGB	Detekovaná barva (0=nice, 1=red, 2=green,3=blue)		22
Stav bezpečné pozice	ano/ne (1=ano)	6	
Akce	0=nic, 1=přesun, 2=sání, 3=rgb, 4=dopravník, 5=kolejnice, 6=home		23



Obr. 4.1: Blokové schéma finální realizace komunikace

Porovnání prvotního návrhu a finální realizace lze vidět v tab. 4.2

Tab. 4.2: Porovnání prvotního návrhu a finální realizace mobilní aplikace

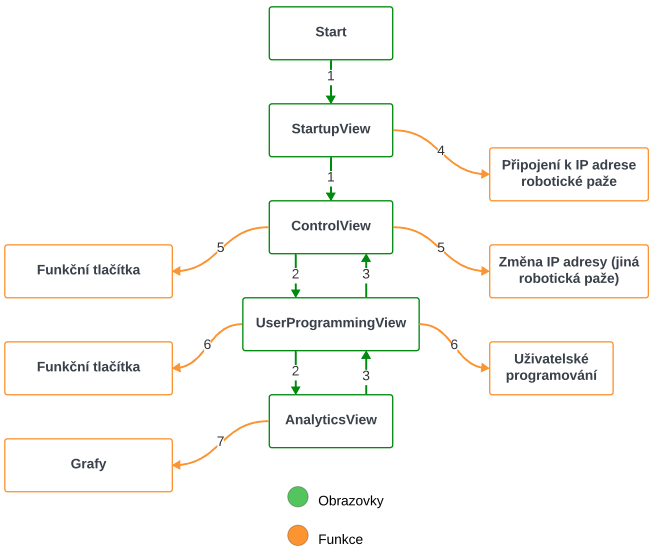
	Prvotní návrh realizace	Finální návrh realizace
Zpoždění	Větší (z důvodu většího množství mezikroků: HTTP protokol, Python skripty, Flask server)	Menší (pouze se zapíší data do registrů či coilů a spustí se robotická paže)
Vývoj	Náročný (je potřeba pracovat s větším množstvím proměnných, s různými programovacími jazyky apod.)	Jednodušší (kód je v rámci jednoho programovacího jazyka, logika a vše potřebné je na jednom místě)
Chyby a reakce na ně	Náročné (viz. Zpoždění)	Jednodušší (pokud jsou všechny části kódu dobře ošetřené, tak je opravdu velmi jednoduché přijít na chybu a opravit ji během chvíle)
Implementace nových funkcí	Náročné (pokud se přidá nová funkce do aplikace, je potřeba ji přidat na Flask server a taktéž jako skript v pythonu)	Jednodušší (je potřeba pouze znát rozložení a parametry registrů a coilů, dále je přidávání dalších funkcí pouze otázka možností dané robotické paže)

4.1 Popis realizace finální komunikace

- **Uživatel** – uživatel je realizován fyzickou osobou, která pracuje s mobilní aplikací (**2**).
- **Mobilní aplikace** (Flutter) – naprogramovaná aplikace, která generuje protokol Modbus/TCP a komunikuje s Raspberry Pi a ovládá robotickou paži. (**3**).
- **Raspberry Pi** – běží zde Modbus server, který přijímá Modbus/TCP protokol a řídí robotickou paži.
- **Robotická paže** – realizuje výsledný pohyb a vrací stavové informace (**4**).

4.2 Realizace finální mobilní aplikace

Pro realizaci finální komunikace bylo potřeba modifikovat základní mobilní aplikaci, viz obr. 4.2.



Obr. 4.2: Realizace finální mobilní aplikace Flutter

4.3 Popis finální mobilní aplikace

- **Start a StartupView** – uživatel spustí, pomocí ikony na ploše zařízení, aplikaci. Provede se metoda main(), která inicializuje Modbus službu, tedy vytvoří Modbus klienta na dané IP adrese a portu a pokusí se navázat komunikaci s robotickou paží. Tato akce se provádí při spuštění aplikace, kde uživatel vidí „uvítací“ obrazovku StartupView viz obr. 4.3.
- **ControlView** – po úspěšné, či neúspěšné inicializaci Modbus služby se objeví místo StartupView obrazovka ControlView. Na této obrazovce je vidět název aplikace, port, který je využíván, a informace, zda se podařilo připojit k Modbus serveru. Dále je zde možnost pro změnu IP adresy a následného připojení (pokud by chtěl uživatel navázat komunikaci s jínou robotickou paží). Nakonec jsou zde různé funkční tlačítka, pro otestování komunikace, jako je například tlačítko pro *Bezpečnou pozici*, které při kliknutí zapíše do registrů konkrétní hodnoty a spustí přesun robotické paže o tyto souřadnice viz obr. 4.4.
- **UserProgrammingView** – pomocí horního navigačního panelu, který je vidět napříč celou aplikací, je možné se dostat na další obrazovky, mezi které patří UserProgrammingView. Tato obrazovka slouží primárně pro uživatelské programování. Je zde připraveno pole, které obsahuje různé tlačítka (i již zmíněné tlačítko *Bezpečná pozice*). Tato tlačítka lze přemístit, dle libosti do dalšího, akčního, pole, které se nachází nad ním. Všechny tlačítka, se postupně ukládají vedle sebe, což udává, v jakém pořadí se budou dané akce, které uživatel nadefinoval, spouštět. Součástí akčního pole je i menší tlačítko, které slouží pro cyklení daných funkcí v akčním poli. Uživatel tedy nejdříve přemístí do akčního pole funkce, které chce, aby robotická paže provedla. Poté nastaví, díky uživatelskému programování a opakujícího se cyklu, kolikrát má tyto úkony paže provést a poté vše spustí tlačítkem *Spustit*. Pro případné změny v rámci akčního pole slouží tlačítko *Vymazat akční oblast* viz obr. 4.5.
- **AnalyticsView** – pomocí již zmíněného horního navigačního panelu je možné zobrazit další obrazovku, tedy AnalyticsView, která slouží pro zobrazení grafů různých závislostí viz obr. 4.6. Je možné zobrazovat dobu, za kterou u jednotlivých funkcí dokáže robot přejít z pohyblivého do klidového stavu, nebo rychlost reakce robotické paže na příkazy od uživatele.

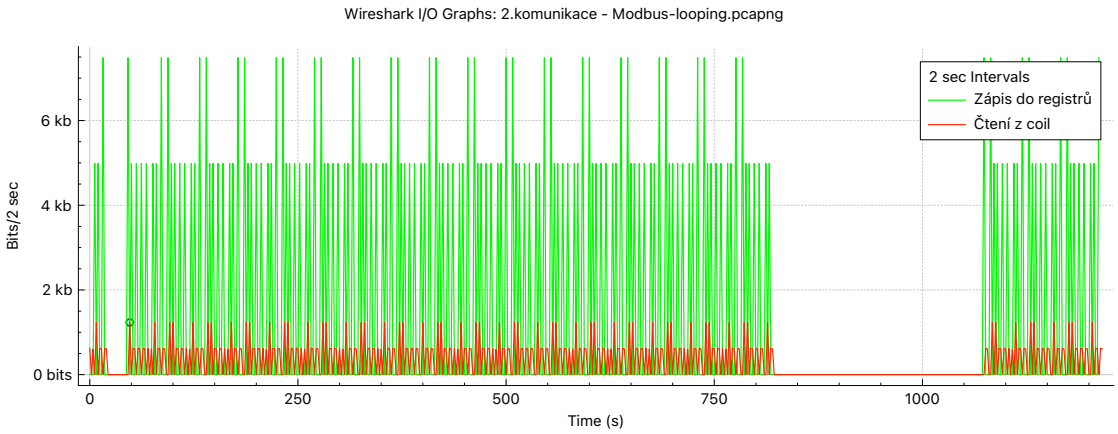
Obr. 4.3: StrartupView

Obr. 4.4: ControlView

Obr. 4.5: UserProgrammingView

Obr. 4.6: AnalyticsView

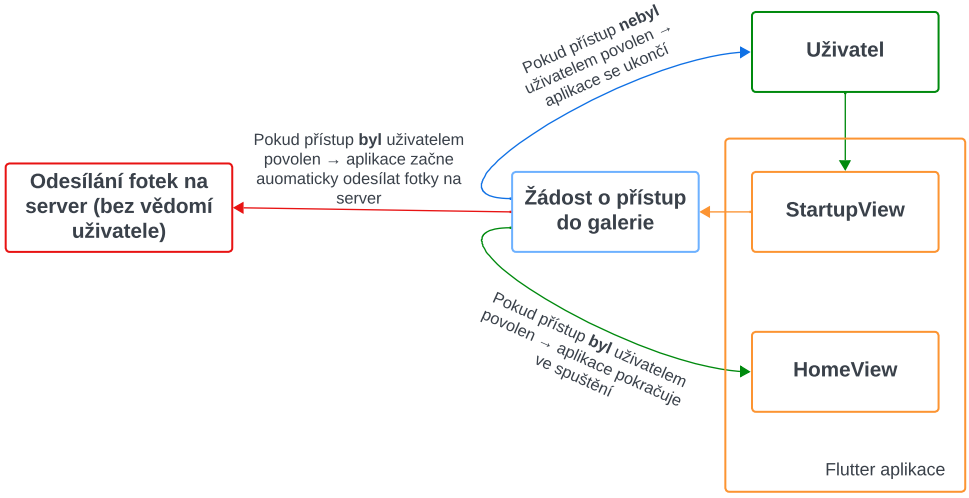
Vizualizace přenesených dat pomocí nástroje Wireshark



Obr. 4.7: Vizualizace přenesených dat pomocí nástroje Wireshark

4.4 Demonstrace zranitelnosti mobilních aplikací

Je důležité taktéž poukázat na zranitelnost mobilních aplikací. Pomocí OWASP (Open Web Application Security Project) byly vyvozeny základní bezpečnostní nedostatky mobilních aplikací. Konkrétně bylo využito a demonstrováno pomocí další naprogramované – „neoprávněné získávání dat pomocí mobilní aplikace“ viz obr. 4.8 [32].



Obr. 4.8: Návrh demonstrace neoprávněného získávání dat

- **Uživatel** – reprezentuje fyzickou osobu, která spouští mobilní aplikaci
- **StartupView** – obrazovka, která se spustí pro „uvítání“ uživatele
- **Žádost o přístup do galerie** – v podobě vyskakovacího dialogového okna aplikace žádá o přístup ke galerii uživatele
- **Odesílání fotek na server a HomeView** – v případě, že uživtel povolí přístup ke galerii, tak se začnou posílat fotky na server, který běží na RaspberryPi. Uživatel avšak o tom neví, vidí pouze spuštěnou aplikaci, konkrétně obrazovku HomeView

Scénář demonstrace neoprávněného získávání dat

1. Ve firmě se rozšíří e-mailem mezi zaměstnance nová aplikace. Jejím účelem je:
   - přidávání zlepšovacích nápadů
   - focení závad
   - focení bezpečnostních rizik

   Za dobrý nápad, či důležitou závadu je nabízena finanční odměna.
2. Zaměstnanci aplikaci důvěřují, jelikož byla údajně poskytnuta IT oddělením. Finanční odměna zaměstnance láká a tak si aplikaci nainstalují.
3. Aplikace při spuštění požádá zaměstnance o přístup ke galerii, což zaměstnance nijak nepřekvapí, jelikož je aplikace určena také pro focení.
4. Avšak jakmile zaměstnanec povolí přístup do galerie, tak se začnou všechny fotky z galerie uživatele posílat na vzdálený server, kde se dále zpracovávají.
5. Uživatel o této skutečnosti nic neví. Vidí pouze spuštěnou aplikaci, která vypadá a slouží tak, jak byla popsána v prvotním e-mailu.
6. Po nějaké době se aplikace dostane až k IT oddělení firmy, které potvrdí, že žádnou aplikaci nedistribuovalo, což vede k závěru, že byla rozšířena podvodná aplikace a bylo neoprávněně zacházeno s daty zaměstnanců.
7. Mezi dopady vyplývající z této skutečnost patří:
   - porušení GDPR
   - ztráta důvěry zaměstnanců vůči firmě
   - bezpečnostní riziko firmy

Závěr

V rámci této bakalářské práce byla vyvinuta snaha o podrobný popis průmyslových sítí a jejich náležitostí, jako jsou komunikační modely ISO/OSI a TCP/IP. Dále zde byly podrobně popsány a porovnány dva velmi používané průmyslové protokoly – Modbus a Profibus. V další části byly vylíčeny mobilní aplikace z hlediska využitelnosti, vývoje a možnosti komunikace s průmyslovými protokoly. Byly zde popsány nástroje pro správu a vývoj mobilních aplikací a porovnány frameworky Flutter a React.

V semestrální praktické části byla zrealizována prvotní komunikace mezi mobilní aplikací v mobilním telefonu a robotickou paží. Byl vytvořen návrh prvotní komunikace, mobilní aplikace Flutter a serveru Flask v podobě blokového schématu. Byly podrobně vylíčeny všechny důležité části blokového schématu, které se podílely na realizaci komunikace mezi mobilní aplikací a robotickou paží. Veškerý zdrojový kód byl vložen do systému Git a spravován pomocí cloudové platformy GitHub a aplikace GitHub Desktop. Pro psaní a ladění zdrojového kódu bylo využito integrované vývojové prostředí Android Studio a nativně podporovaného emulátoru, který reprezentoval mobilní telefon. Díky mobilní aplikaci byla možnost, pomocí dotykových tlačítek, ovládat robotickou paži.

V bakalářské praktické části bylo navázáno na prvotní návrh a následně byla provedena optimalizace za účelem zefektivnění provozu snížením celkového zpoždění komunikace. Dále byl otestován způsob komunikace mezi mobilní aplikací a robotickou paží. Díky tomu bylo možné vynechat Flask server a skripty, což způsobilo menší zpoždění, lepší vývoj, řízení, opravování a hledání chyb, způsobených reálnými testy, a vyšší variabilitu samotné aplikace. Byl implementován externí Modbus balíček. Dále byla vytvořena globální metoda pro připojení a odpojení k Modbus/TCP serveru, zápis do registrů a čtení z registrů. Funkce, které byly vytvořeny v této globální metodě, se využívají napříč celou aplikací. Byly přidány nové obrazovky, konkrétně uživatelské programování, vizualizace stavu registrů a připojení. V neposlední řadě byla vytvořena obrazovka pro analytiku. Byla přidána i funkce pro změnu IP adresy, díky které se lze v rámci aplikace přepojovat a ovládat další robotické paže. Pomocí programu Wireshark byla podrobně otestována reálná komunikace mezi mobilní aplikací a robotickou paží.

Výsledkem práce je funkční mobilní aplikace a realizovaná komunikace s robotickou paží. Autor práce věří, že mobilní aplikace mají velký, ještě nevyužitý potenciál, a že s jejich pomocí lze dosti zjednodušit a zefektivnit správu a kontrolu průmyslových zařízení.

Literatura

[1] Brendan Galloway and Gerhard P. Hancke, Senior Member, IEEE. *Introduction to Industrial Control Networks*. Online. Dostupné z: https://www.rfidblog.org.uk/Preprint-GallowayHancke-IndustrialControlSurvey.pdf. [cit. 2024-11-28].

[2] Keith Stouffer; Michael Pease; CheeYee Tang; Timothy Zimmerman; Victoria Pillitteri; Suzanne Lightman; Adam Hahn; Stephanie Saravia; Aslam Sherule; Michael Thompson. *Introduction to Industrial Control Networks*. Online 2023. Dostupné z: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-82r2.pdf. [cit. 2024-12-07].

[3] Schneider Electric. *Indutrial networks*. Online 2024. Dostupné z: http://at.dii.unipd.it/renato.gobbo/didattica/corsi/Componenti_tecnologie_elettrici/documenti_schneider/automation_guide/asg-9-industrial-networks.pdf. [cit. 2024-12-07].

[4] Palo Alto Networks. *What Is the Difference Between IT and OT? | IT vs. OT*. Online. 2024. Dostupné z: https://www.paloaltonetworks.com/cyberpedia/it-vs-ot. [cit. 2024-11-28].

[5] Palo Alto Networks. *Industrial protocols*. Online. 2024. Dostupné z: https://www.bostontech.net/wp-content/uploads/2020/10/Practical-Data-Communications-and-Networking-9.Industrial-protocols.pdf. [cit. 2024-11-28].

[6] Michael J. Hamill, PE. *Industrial Communications and Control Protocols*. Online. 2020. Dostupné z: https://pdhonline.com/courses/e497/e497content.pdf. [cit. 2024-11-28].

[7] Eric Knapp; Science Direct. *Industrial Network Protocol*. Online. 2011. Dostupné z: https://doi.org/10.1016/B978-1-59749-645-2.00001-X. [cit. 2024-11-28].

[8] Shandong Renke Control Technology Co.,Ltd. *What is Modbus Protocol and its Types?*. Online. 2024. Dostupné z: https://www.renkeer.com/modbus-protocol-and-its-types/. [cit. 2024-11-28].

[9] Kevin Lamshöft; Jana Dittmann. *Assessment of Hidden Channel Attacks: Targetting Modbus/TCP*. Online. 2020. Dostupné z: https://doi.org/10.1016/j.ifacol.2020.12.258. [cit. 2024-11-28].

[10] Ricardo J. Rodríguez; Stefano Marrone; Ibai Marcos; Giuseppe Porzio. *MOSTO: A toolkit to facilitate security auditing of ICS devices using Modbus/TCP*. Online. 2023. Dostupné z: https://doi.org/10.1016/j.cose.2023.103373. [cit. 2024-11-29].

[11] Chen, Bo and Pattanaik, Nishant and Goulart, Ana and Butler-purry, Karen L. and Kundur, Deepa. *Implementing attacks for modbus/TCP protocol in a real-time cyber physical system test bed*. Online. 2015. Dostupné z: https://ieeexplore.ieee.org/abstract/document/7129084?casa_token=fnFcCAcp828AAAAA:Qh1NXjpNg9oHDBVNUFdjCva7P7E_lusZ7eyNBO7HY6M1LL83TQPu19HA9Iqcv-2AFPRvM4LX7fyq. [cit. 2024-12-07].

[12] Irfan A. Siddavatam; Sachin Parekh; Tanay Shah; Faruk Kazi. *Testing and Validation of Modbus/TCP Protocol for Secure SCADA Communication in CPS using Formal Methods*. Online. 2017. Dostupné z: https://scpe.org/index.php/scpe/article/view/1331. [cit. 2024-12-07].

[13] Nardone, Roberto and Rodríguez, Ricardo J. and Marrone, Stefano. *Formal security assessment of Modbus protocol*. Online. 2016. Dostupné z: https://ieeexplore.ieee.org/document/7856685. [cit. 2024-12-07].

[14] Dai-Rong, Yu and Fei, Bian and Bo, Zhang. *Improving TLS Protocol Using Identity-Based Double-certificate Mechanism*. Online. 2012. Dostupné z: https://ieeexplore.ieee.org/document/6322310. [cit. 2024-12-07].

[15] Modbus Organization, Inc. *MODBUS/TCP Security*. Online. 2021. Dostupné z: https://www.modbus.org/docs/MB-TCP-Security-v36_2021-07-30.pdf. [cit. 2024-12-07].

[16] Luís Ferreira; Eduardo Tovar. *THE INFLUENCE OF INTER-DOMAIN MOBILITY ON MESSAGE STREAM RESPONSE TIME IN WIRED/WIRELESS PROFIBUS-BASED NETWORKS Author links open overlay panel* . Online. 2005. Dostupné z: https://doi.org/10.3182/20051114-2-MX-3901.00020. [cit. 2024-11-29].

[17] Hongjun Chen; Xiaohua Zbang; Xuerui Zbang. *Applications of Profibus to Industrial Automation*. Online. 1998. Dostupné z: https://doi.org/10.1016/S1474-6670(17)36378-4. [cit. 2024-11-29].

[18] *ProfiBus overview*. Online. Dostupné z: https://home.agh.edu.pl/~ipnet/Materials1/Module2/ProfiBus_overview.pdf. [cit. 2024-11-29].

[19] Viral Nagda. *Difference Between Modbus and Profibus*. Online. 2024. Dostupné z: https://instrumentationtools.com/difference-between-modbus-and-profibus/#google_vignette. [cit. 2024-11-29].

[20] Katie Terrell Hanna, Ivy Wigmore. *mobile app*. Online. 2023. Dostupné z: https://www.techtarget.com/whatis/definition/mobile-app. [cit. 2024-12-8].

[21] Coursera Staff. *What Is UI Design? Definition, Tips, Best Practices* . Online. 2024. Dostupné z: https://www.coursera.org/articles/ui-design. [cit. 2024-11-29].

[22] Digidop. *Figma vs Penpot comparison*. Online. 2024. Dostupné z: https://www.digidop.fr/en/figma-vs/penpot. [cit. 2024-11-29].

[23] Technostacks. *Most Popular Mobile App Development Frameworks For App Developers*. Online. 2024. Dostupné z: https://technostacks.com/blog/mobile-app-development-frameworks/. [cit. 2024-11-29].

[24] Flutter. *Build for any screen*. Online. 2024. Dostupné z: https://flutter.dev/. [cit. 2024-11-29].

[25] React. *The library for web and native user interfaces*. Online. 2024. Dostupné z: https://react.dev/. [cit. 2024-11-29].

[26] W3Schools. *React Tutorial*. Online. 2024. Dostupné z: https://www.w3schools.com/react/default.asp. [cit. 2024-11-29].

[27] Miquido. *Flutter vs React Native: Which one to choose for your mobile app in 2024*. Online. 2024. Dostupné z: https://www.miquido.com/blog/flutter-vs-react-native-comparison/. [cit. 2024-11-29].

[28] Google; Flutter; Dart. *The official package repository for Dart and Flutter apps.*. Online. 2024. Dostupné z: https://pub.dev/. [cit. 2024-11-29].

[29] GitHub, Inc. *About GitHub and Git*. Online. 2024. Dostupné z: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git. [cit. 2024-11-29].

[30] Kymberley R. Scroggie; Klementine J. Burrell-Sander; Peter J. Rutledge; Alice Motion. *GitHub as an open electronic laboratory notebook for real-time sharing of knowledge and collaboration*. Online. 2023. Dostupné z: https://doi.org/10.1039/d3dd00032j. [cit. 2024-11-29].

[31] Pallets. *A Minimal Application*. Online. 2010. Dostupné z: `https://flask.palletsprojects.com/en/stable/quickstart/#http-methods`. [cit. 2024-11-29].

[32] Mark Curphey. *OWASP Top Ten 2025*. Online. 2025. Dostupné z: `https://owasp.org/www-project-top-ten/#`. [cit. 2025-06-02].