VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ KNIHOVNA GRAFICKÝCH KOMPONENT PRO VIZUALIZACI DAT Z CHYTRÝCH ZAŘÍZENÍ

WEB-BASED LIBRARY OF GRAPHICAL COMPONENTS FOR SMART DEVICES DATA VISUALIZATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ONDŘEJ DARMOPIL**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JIŘÍ HYNEK, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

154289

Ústav: Ústav informačních systémů (UIFS)
Student: **Darmopil Ondřej**
Program: Informační technologie
Název: **Webová knihovna grafických komponent pro vizualizaci dat z chytrých zařízení**
Kategorie: Informační systémy
Akademický rok: 2023/24

Zadání:

1. Prostudujte problematiku internetu věcí (*Internet of Things*, IoT) a chytrých měst (*Smart City*).
2. Nastudujte principy vizualizace dat a tvorby uživatelských rozhraní informačních systémů a znovupoužitelných grafických komponent.
3. Analyzujte požadavky firmy Logimic na vizualizaci dat z chytrých zařízení a grafické komponenty vhodné pro tento účel.
4. Dle výsledků analýzy navrhněte knihovnu znovupoužitelných responzivních grafických komponent, určených pro vizualizaci dat z chytrých zařízení
5. Navrženou knihovnu grafických komponent implementujte a integrujte do platformy Smart City firmy Logimic.
6. Otestujte použitelnost a škálovatelnost navržených grafických komponent.

Literatura:

- Greengard, S. (2015). *The Internet of Things*. MIT Press.
- Kirimtat, A., Krejcar, O., Kertesz, A., & Tasgetiren, M. F. (2020). Future trends and current state of smart city concepts: A survey. *IEEE access*, *8*.
- Interní dokumentace firmy Logimic.

Při obhajobě semestrální části projektu je požadováno:
Body 1 - 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hynek Jiří, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024
Datum schválení: 30.10.2023

Abstrakt

Cílem této bakalářské práce je analyzovat již existující grafické komponenty webové aplikace ACADA. Jedná se o aplikaci vyvíjenou společností Logimic. Tato aplikace má za úkol správu jednotlivých chytrých zařízení a následnou vizualizaci nasbíraných dat. Na základě provedené analýzy poté navrhnout jednotnou grafickou knihovnu, která bude obsahovat základní stavební prvky každé webové aplikace, ale také komponenty pro zobrazování dat ze zařízení. Knihovna se zaměří na konfigurovatelnost aplikace z pohledu uživatele. V rámci implementace v jazyce Angular bude kladen důraz na dynamické vykreslování komponent.

Abstract

The objective of this bachelor's thesis is to analyze the existing graphical components of the ACADA web application. ACADA is an application developed by Logimic that manages individual smart devices and visualizes the collected data. Based on this analysis, a unified graphical library will be designed that will include the basic building blocks of any web application, as well as components for displaying data from the devices. The library will focus on the configurability of the application from the user's perspective. In the context of the implementation in Angular, the focus will be on dynamic component rendering.

Klíčová slova

IoT, Chytré město, UX, UI, grafická knihovna, znovupoužitelné komponenty

Keywords

IoT, Smart city, UX, UI, graphic library, reusable components

Citace

DARMOPIL, Ondřej. *Webová knihovna grafických komponent pro vizualizaci dat z chytrých zařízení.*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Hynek, Ph.D.

Webová knihovna grafických komponent pro vizualizaci dat z chytrých zařízení

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Hynka, Ph.D. Další informace mi poskytla firma Logimic. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Ondřej Darmopil
8. května 2024

Poděkování

Tímto bych chtěl poděkovat Ing. Jiřímu Hynkovi, Ph.D. za umožnění této bakalářské práce, velmi časté konzultace a jeho vedení při této práci. Dále bych chtěl poděkovat Markovi Konečnému a firmě Logimic za zpětnou vazbu.

Obsah

Kapitola 1

Úvod

S lepší dostupností internetu pro širokou veřejnost se začalo více používat chytrých zařízení, která se připojovala do sítě. Tímto se začal rozšiřovat pojem internetu věcí (IoT). Jedná se o síť propojených zařízení, která odesílají data na hlavní centrální místo, kde se data shromažďují, analyzují a následně zobrazují. Tato zařízení mají velmi vysokou energetickou efektivitu z důvodu toho, aby se dosáhlo co nejdelší výdrže na baterie. Síť internetu věcí se využívá ve velkých organizacích například ve firmách. Další důležitou oblastí internetu věcí se stala správa měst, která vyplynula z potřeby spravovat jednotlivá odvětví města. V důsledku této expanze byla potřeba vytvářet různé systémy, které zajišťují analýzu sesbíraných dat do větších datových celků.

Jednou ze společností, která vyvíjí takovéto systémy, je firma Logimic[1]. Jedná se o firmu vyvíjející cloudovou aplikaci ACADA, která umožňuje komunikaci s různými IoT zařízeními. ACADA je konfigurovatelná platforma pro připojení všech potřebných IoT zařízení. Je možné si nakonfigurovat ukládaná data, zobrazovaná data a jejich formát. V rámci aplikace ACADA neexistuje jednotná grafická knihovna s komponentami. To nejen ztěžuje práci vývojářům při rozšiřování aplikace, ale také nepůsobí odborným dojmem, když je každá komponenta jiná (vzhledově). Dále je ze strany uživatelů vyžadována větší přizpůsobitelnost.

Zaměřením této bakalářské práce bude sjednotit grafickou knihovnu s cílem usnadnit vývojářům tvorbu konzistentních aplikací. Dále tuto knihovnu rozšířit o komponenty, které umožní uživatelskou přizpůsobitelnost. Kapitola 2 vysvětluje termín internet věcí, jeho základní třívrstvý model a rozšířený pětivrstvý model, jejich porovnání a popis jednotlivých vrstev. Také popisuje topologie používané v internetu věcí. Poslední součástí kapitoly je pojem chytré město. Také obsahuje definici řídícího panelu chytrého města, oblasti využití chytrého města a konkurenční aplikace pro správu. Kapitola 3 vysvětluje vizualizaci dat. Dále popisuje základní tvorbu uživatelských rozhraní a její principy. Poslední část kapitoly se věnuje znovupoužitelným grafickým komponentám a principům jejich tvorby. Na základě těchto znalostí proběhne analýza 4 již existujících komponent v aplikaci ACADA a návrh 5 nové grafické knihovny, která pokryje potřeby internetu věcí. Nakonec dojde k implementaci této knihovny 6 a jejímu otestování 7 pomocí již existující aplikace ACADA.

[1] https://www.logimic.com/

Kapitola 2

Internet věcí a chytré město

Internet věcí a chytré město jsou pojmy, které se v posledních letech velmi rozšiřují. V této kapitole je rozebrán pojem internet věcí, jeho základní model a rozšířený model. Také je zde vysvětlena komunikace v IoT a její typy pro dnešní použití. Dále se tato kapitola zajímá o termín chytré město a jeho závislost na IoT. Další důležitý termín je KPI (klíčový ukazatel výkonnosti) a jeho účel pro chytrá města.

2.1 Internet věcí

Termín *Internet věcí* (anglicky *Internet of things*, IoT) byl podle článku [22] použit již v roce 1999 jako systém, v němž by objekty ve fyzickém světě mohly být připojeny k internetu pomocí radiofrekvenční identifikace (RFID). V roce 2014 bylo časopisem [26] IoT definováno jako paradigma, které představuje blízkou budoucnost, kde budou běžné předměty vybaveny mikrokontroléry, vysílači pro digitální komunikaci a vhodnými protokolovými vrstvami. Tyto vrstvy jim umožní komunikovat mezi sebou a s uživateli, stávajíce se nedílnou součástí internetu. Dle knihy [17] označuje termín IoT vestavná zařízení (*věci*), která mají internetovou konektivitu umožňující vzájemnou komunikaci s ostatními zařízeními a lidmi v globálním měřítku. Podrobnější definice popisuje IoT jako svět, kde jsou fyzické objekty bez problémů integrovány do informační sítě a mohou se stát aktivními účastníky obchodních procesů. Služby jsou schopné komunikovat s těmito „chytrými objekty“ přes internet, dotazovat se na jejich stav a veškeré jejich informace s ohledem na bezpečnost [25]. Tato síť zařízení se využívá ve firmách, ale také ve městech pro transformaci na *Chytré město* 2.3.

2.1.1 Základní model internetu věcí

Nejznámějším modelem IoT je architektura se třemi vrstvami skládající se z vrstvy vnímání (*perception layer*), síťové vrstvy (*network layer*) a aplikační vrstvy (*application layer*) [19], jak je vidět na obrázku 2.1.

- **Vrstva vnímání:** se skládá z objektů/zařízení a senzorů, které zabezpečují sběr dat a informací [19]. V článku [24] je vrstva vnímání popsána jako skupina zařízení s připojením k internetu a tato zařízení jsou schopna vnímat, detekovat objekty, shromažďovat a vyměňovat si informace s ostatními zařízeními prostřednictvím internetových komunikačních sítí. Zahrnuje to zařízení pro radiofrekvenční identifikaci (RFID), kamery, senzory a systémy globálního určení polohy (GPS).

- **Síťová vrstva:** se používá pro přenos a zpracování dat poskytovaných vrstvou vnímání. Funguje jako nervová síť a mozek v lidském těle [19]. Technologie pracující na této vrstvě zahrnují 3G, WiFi, Bluetooth, infračervené záření, Zigbee a další.
- **Aplikační vrstva:** se na našem obrázku nachází na posledním místě. Je zodpovědná za obdržení a zpracování informací [24].

Obrázek 2.1: Základní model versus pětivrstvý model – inspirován [19]

2.1.2 Rozšířený model IoT

Základní model 2.1.1 nebere v úvahu rozšiřování technologií každý rok a spolu s ní potřebu správných metod a obchodních modelů, proto je potřeba použití nové architektury, která představuje rozšířený model pro IoT vytvořený kombinací modelu TCP/IP a komunikačního modelu s požadavky na IoT [19]. Rozšířený model obsahuje navíc middleware a obchodní vrstvu, jak je znázorněno na obrázku 2.1.

- **Middleware vrstva:** je zodpovědná za správu služeb a má spojení s databází, do které ukládá data přijata ze síťové vrstvy [12]. Propojuje žadatele o službu s danou službou, což umožňuje interakci heterogenních zařízení bez ohledu na hardwarovou platformu [19].
- **Obchodní vrstva:** je zodpovědná za správu celkového IoT, pro které vytváří obchodní modely s hodnocením výkonnosti existujících aplikací a služeb z dat získaných od aplikační vrstvy [12, 19]. Na základě těchto hodnocení pomáhá tato vrstva určit budoucí akce a obchodní strategie [12].

2.2 Komunikace v IoT

Podle článku [4] se obecně v IoT používají dvě kategorie sítí. Krátký dosah a dlouhý dosah, ale vše s důrazem na nízkou spotřebu energie. Sítím s krátkým dosahem se také někdy říká *posledních 100 metrů připojení.*

V článku [7] se pro krátký dosah využívají technologie WLAN[1], Bluetooth, ZigBee. Pro větší dosah se využívají celulární sítě (2G, 3G), které jsou ale vytvořené pro přenos mobilní komunikace (hlasu, videa a dat), což má velké náklady na spotřebu energie. Jako reakce na potřeby IoT se vyvinula technologie LPWAN.

[1] bezdrátová lokální síť, anglicky wireless local area networks

2.2.1 LPWAN komunikace

Nárůst koncových zařízení např. senzorů v IoT si vyžádal vytvoření nových komunikačních standardů. Tím vzrostlo použití komunikace LPWAN, která tyto zařízení připojí k síti. LPWAN (anglicky *low-power wide area network*) je třída bezdrátových komunikačních standardů IoT, která řeší pokrytí velké plochy při malé náročnosti na baterii za využití malé přenosové rychlosti [7].

V článku [15] se zmiňují tři největší technologie (popsané níže v seznamu) využívající LPWAN komunikaci. V tabulce 2.1 je srovnání základních vlastností těchto technologií.

- **Sigfox:** je provozovatelem LPWAN sítě nabízející komplexní řešení internetu věcí pomocí jeho patentovaných technologií. Jsou využívána nelicencovaná pásma. V České republice fungoval na frekvenci 868 MHz [15]. Výhody řešení Sigfox jsou například nízká spotřeba energie, vysoké citlivosti přijímače a levná konstrukce antény. Nevýhodou je poté nízká rychlost přenosu (100 bit/s) a nemožnost posílat aktualizace na zařízení. Ze strany Sigfoxu je komunikace omezena denně na 144 zpráv [15].

- **LoRaWAN:** (anglicky Long Range Wide Area Network) je volně přístupná technologie, na které mohou společnosti stavět vlastní řešení. LoRaWAN využívá radiovou komunikaci na nelicencovaných pásmech. Pro sítě tohoto typu jsou dostupné rychlosti od 300 bit/s do 50 000 bit/s. Z důvodu volně přístupné technologie je možné v těchto IoT sítích používat zařízení různých výrobců. Další nespornou výhodou je fakt, že můžeme koncová zařízení aktualizovat vzdáleně přes obousměrnou komunikaci [2].

- **NB-IoT:** je úzkopásmová technologie internetu věcí. NB-IoT pracuje s globálním systémem pro mobilní komunikaci (GSM) a LTE technologií na licencovaných frekvenčních pásmech. Komunikační protokol NB-IoT vychází z protokolu LTE, ale omezuje počet odeslaných zpráv na pouze ty nejdůležitější. Výhoda tohoto řešení je, že není nutné budovat infrastrukturu, ale je možné použít již stávající. Oproti předchozím řešením je toto náročné na spotřebu energie [15].

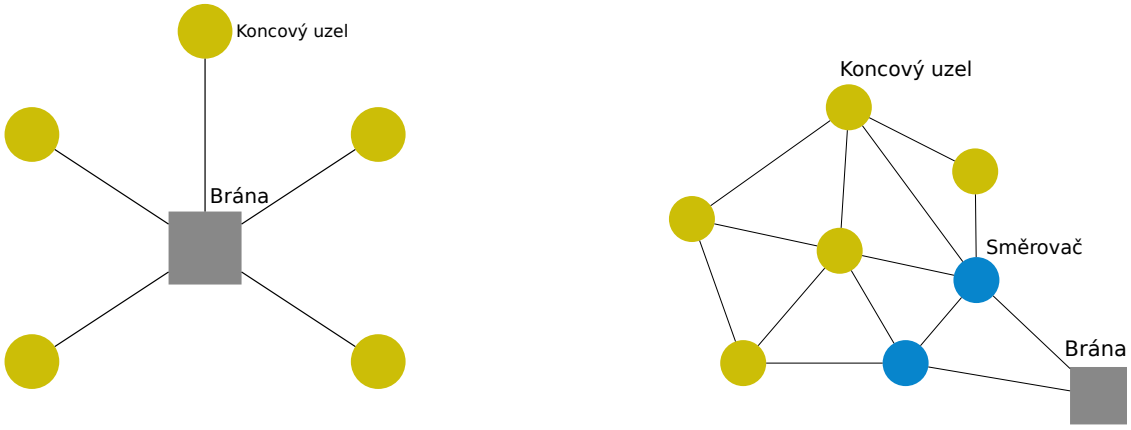
	Sigfox	**LoRaWAN**	**NB-IoT**
Frekvence	Nelicencované pásmo	Nelicencované pásmo	Licencované LTE pásmo
Šířka pásma	100 Hz	250 kHz a 125 kHz	200 kHz
Maximální přenosová rychlost	100 bit/s	50 kbit/s	200 kbit/s
Omezení zpráv za den	144 zpráv	Neomezeno	Neomezeno
Standardizace	Firma Sigfox	LoRa-Alliance	3GPP

Tabulka 2.1: Srovnání dostupných LPWAN technologií - inspirováno z článku [15]

Topologie sítí

Tato podsekce je převzata z článku [7]. Dvě významné topologie využívající se v sítích LPWAN jsou hvězda (*star*) a *mesh*[2]. Nejjednodušší způsob komunikace je, když koncová

[2] v češtině se pro daný anglický název nevyskytuje vhodné slovíčko



(a) Topologie hvězda

(b) Mesh topologie

Obrázek 2.2: Porovnání hvězdy a mesh topologie - inspirováno článkem [7]

zařízení přímo komunikují s bránou (*gateway*) přesně jako topologie hvězda, kterou můžeme vidět na obrázku 2.2a. Také tato komunikace nabízí značnou úsporu baterií a větší dosah z důvodu jediného skoku na bránu. Pro komunikaci mezi zařízeními musí být využita brána. Koncová zařízení mohou odesílat data na více bran. Poté data putují na centrální server, kde se kontrolují duplikáty, chyby a provádí se bezpečnostní kontroly. Při nefunkčnosti koncového uzlu je možné toto zařízení identifikovat. Při selhání brány se stanou všechny koncové uzly nedostupnými.

Topologie mesh, kterou můžeme vidět na obrázku 2.2b, obsahuje navíc směrovací uzly (*routery*). Také se zde nacházejí nadbytečné cesty mezi uzly, které zabezpečují více tras pro dosažení cíle. Topologie mesh je lehce rozšiřitelná. Nevýhody těchto sítí jsou: příliš velká složitost při větších sítích, větší odezva sítě, větší náklady na stavbu sítě a také na spotřebu energie.

2.3 Chytré město

Chytré město (anglicky *Smart City*) je složitý ekosystém charakterizovaný intenzivním využitím informačních a komunikačních technologií (ICT) s cílem učinit města atraktivnějšími, udržitelnými, jedinečnými místy pro inovace a podnikání [14].

Zdroj [16] popisuje chytré město jako „koncept, který nemá stále konzistentní a jasnou definici mezi akademiky a praktiky“. Prezentuje dále formální definici chytrého města jako: „město propojující fyzickou, informační, technologickou, sociální a obchodní infrastrukturu k využití kolektivní inteligence města“. Dále definuje chytré město jako: „město, které využívá informační a komunikační technologie (ICT) a jiné prostředky ke zlepšení kvality života, efektivnosti urbanistických operací a služeb a konkurenceschopnosti, přičemž zároveň zajistí splnění potřeb současných i budoucích generací v ekonomických, sociálních a environmentálních aspektech“.

2.3.1 Řídící panel chytrého města

Řídící panel (anglicky *dashboard*) je nástroj, který umožňuje analyzovat, stopovat data a zobrazovat je pomocí *klíčových ukazatelů výkonnosti* podle různých segmentů [6]. Klí-

čový ukazatel výkonnosti (anglicky *Key performance indicator*, zkratka KPI) se v chytrých městech podle článku [21] používá pro sledování a posouzení sociálně-ekonomických a environmentálních dopadů řízení města na jeho „chytrost“. Dále se na základě těchto ukazatelů diagnostikují stávající problémy a identifikují oblasti, které lze zlepšit prostřednictvím lepšího řízení a nových technologických řešení.

V článku [11] se definuje pojem řídící panel jako nástroj, který by měl být schopný načítat starší data, ale také data v reálném čase. Dále by měl ukazovat získaná data pomocí různých grafických a interaktivních vzorů. Tento zdroj ještě definuje body, které by měl dashboard splňovat:

- Je složen z množiny grafických komponent, které mohou být rozmístěny do připravené mřížky.
- Komponenty mohou fungovat samostatně nebo mohou být spojeny s jinými komponenty.
- Skrze komponenty je možné zobrazovat historická data a data v reálném čase, která jsou získávána z použité databáze.
- Komponenty mohou být napojeny na oznamovací systém, který při přesáhnutí určité hodnoty odesílá oznámení uživateli.

2.3.2 Využití chytrých měst pro uživatele

Chytrá města poskytují oboustranně výhodný stav pro občany a veřejnou zprávu. Zdroj [20] popisuje využití IoT v chytrých městech takto:

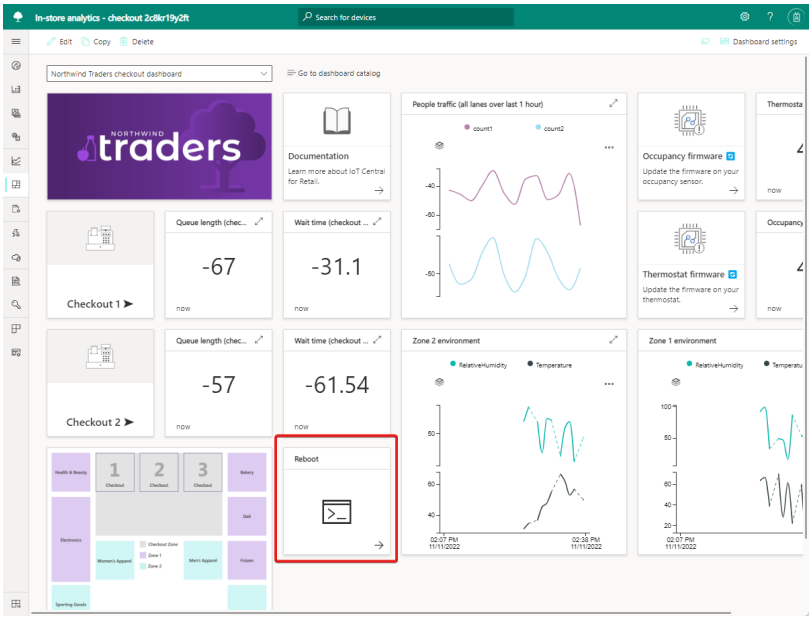
- **Doprava:** ve velkoměstech při rozrůstající se civilizaci zapříčiňuje čím dál tím více nevyhnutelné dopravní zácpy. V chytrém městě se nabízí řešení pomocí chytrých semaforů, které namísto pevně stanoveného časového signálu mění světelnou signalizaci podle aktuálního zaplnění křižovatky. Při použití chytrých semaforů se sníží čekací doba a to má za důsledek snížení spotřeby paliva a emisí.
- **Správa odpadů:** zásadní problém moderních měst je odpadové hospodářství, především kvůli nákladům a místu pro skládky. Pomocí inteligentních kontejnerů na odpad je možné snímat hladinu zaplnění a při zaplnění řídící centrum vyšle svoz, aby nedošlo k přeplnění kontejneru. Také je možné pomocí IoT plánovat optimální trasy svozů pro redukci nákladů na palivo.
- **Kvalita ovzduší:** se každým rokem zhoršuje v důsledku rostoucího množství plynů vypouštěných průmyslem a dopravou ve velkých městech. Na tento problém je možné nasadit chytré senzory, které měří složení plynů v ovzduší a dle těchto senzorů identifikovat nejvíce znečištěné oblasti. Dále se pomocí sesbíraných dat mohou generovat v mobilních aplikacích občanů nejméně znečištěné trasy do práce, obchodů nebo jen pro běhání či chůzi.
- **Osvětlení města:** je ideální zařízení pro shromažďování údajů o světelných podmínkách prostředí, podle kterých se dále rozhodují o zapnutí/vypnutí světel a jeho intenzitě. Díky tomu je možné šetření energie a také je možný vzdálený přístup pro ovládání.

- **Zásobování vodou:** začíná být velmi vzácné, proto je důležité použití chytré správy vody. Vestavěné senzory do potrubí zvládají snímat úniky vody na různých místech. Také snímá průtok konkrétním potrubím a podle toho přiděluje dodávky vody. Systém chytrého hospodaření s vodou poskytuje snížení provozních nákladů, které zahrnují výstavbu, údržbu a další.

2.3.3 Existující aplikace

Na správné používání IoT zařízení jsou vyvíjeny aplikace, do kterých je uživatel schopen si tato zařízení přidat. Přidané zařízení lze následně konfigurovat podle potřeby uživatelů. Dále některé z existujících aplikací na trhu poskytují přizpůsobitelný řídící panel pro zobrazení důležitých stavových dat o zařízení. Skoro každá větší firma vlastní své řešení pro správu IoT zařízení. Firma Google své řešení Cloud IoT Core[3] ukončila.

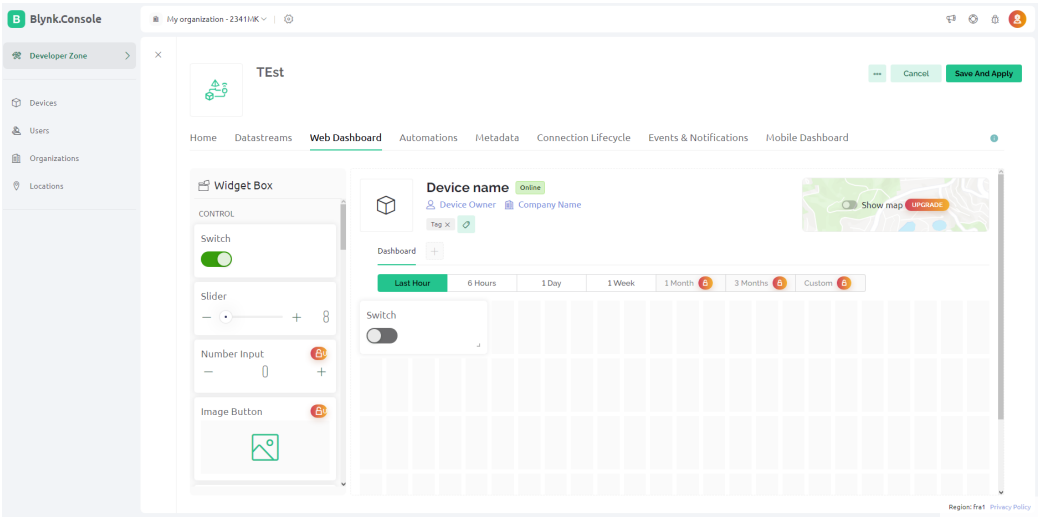
- **Microsoft Azure:** je řešení nabízející firma Microsoft se jmenuje Azure IoT Central[4]. Pro informace o platformě je použita dokumentace Microsoftu [3]. Na platformě Microsoft je možné vytvářet jednotlivé aplikace pro skupiny IoT zařízení. Po vytvoření aplikace a připojení požadovaných IoT zařízení je možné připojené zařízení spravovat. Dále se do platformy ukládají data přijatá ze zařízení. V jednotlivých aplikacích je možné přizpůsobit výchozí řídící panel nebo vytvářet nové řídící panely. Řídící panel v aplikaci Azure IoT Central se skládá ze čtvercových dlaždič, které jsou různě velké. Na řídící panel je možné vkládat vizualizační komponenty, ale také je možné zobrazovat řídící příkazy například pro restartování zařízení, jak je vidět na obrázku 2.3.



Obrázek 2.3: Řídící panel Microsoft Azure - obrázek převzat z dokumentace [3]

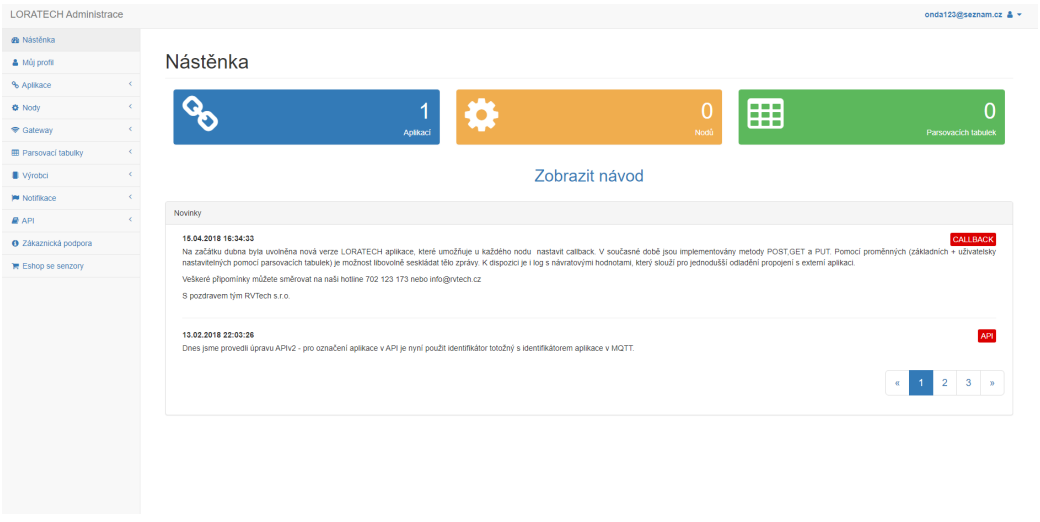
[3] https://cloud.google.com/iot-core
[4] https://azure.microsoft.com/cs-cz/products/iot-central

- **Blynk:** je řešení od firmy Blynk[5] je více zaměřeno na jednotlivá zařízení. V této aplikaci chybí jednotný řídící panel oproti firmě Microsoft. Je ale možné konfigurovat řídící panel pro jednotlivé zařízení zvlášť, jak je možné vidět na obrázku 2.4. Také se toto nastavení dá ukládat do šablon a ty poté použít na nová zařízení. Základní zobrazení aplikace je seznam připojených zařízení.



Obrázek 2.4: Editace řídícího panelu na platformě Blynk

- **Loratech:** je řešení od české firmy RVTech pro chytré obce, města nebo školy se jmenuje Loratech. Podobně jako u firmy Microsoft je možné vytvářet jednotlivé aplikace obsahující IoT zařízení. Data aktuálně odeslané ze zařízení se uchovávají, ale není možné je zobrazit pomocí grafů. Také platforma neobsahuje konfigurovatelný hlavní řídící panel pro zobrazení nejdůležitějších dat, jak je vidět na obrázku 2.5.



Obrázek 2.5: Hlavní panel platformy Loratech

[5] https://blynk.io

Kapitola 3

Vizualizace dat, uživatelské rozhraní a grafické komponenty

Tato kapitola se zabývá vizualizací dat z chytrých měst založených na internetu věcí pro koncové uživatele v sekci 3.1, tvorbou uživatelských rozhraní pro informační systémy v sekci 3.2 a znovupoužitelné grafické komponenty v sekci 3.3.

3.1 Vizualizace dat

Kniha [6] definuje *vizualizaci dat* (anglicky *data visualization*) jako proces, který spotřebovává data jako vstup a transformuje je do poznatků, takže máme data jako surovinu a poznatek jako rafinovaný produkt. Další způsob reprezentace dat nebo informací je pomocí grafu, diagramu nebo další vizuální formáty pro zobrazení poznatků z dat pomocí vizuální reprezentace. Existují dva typy vizualizace dat:

- **Průzkumné vizualizace:** (anglicky *exploratory*) zkoumají data pomocí dotazování a interagování, které je nápomocné k zodpovězení otázek o daných datech. K této vizualizaci je nutný kontext dat. Řeší nejen pochybnosti, ale také vybízí k vytváření nových otázek, které vás zatím nenapadly. Podmnožina průzkumné vizualizace je zobrazení pomocí řídícího panelu.
- **Vysvětlující vizualizace:** (anglicky *explanatory*) zobrazují pouze důležitá data, která jsou statická a v čase neměnná. Příkladem takové vizualizace je infografika. Jedná se o způsob zobrazení dat v takovém grafickém formátu, který je nejlépe čitelný.

3.2 Tvorba uživatelských rozhraní

Článek [23] označuje design a grafiku vším co uživatel na webu vidí, ale kreativita není něco, co by se dalo naučit. Dále mluví o potřebě efektivního zpracování a zobrazení většího množství dat na různých zařízeních. Z toho vyplývá, že uživatelské rozhraní hraje velmi důležitou roli v každé aplikaci. Dále zmiňují potřebu některých firem rozdělit vývoj designu aplikace na uživatelské rozhraní a uživatelskou zkušenost.

3.2.1 Uživatelské rozhraní

Kniha [10] označuje termín uživatelské rozhraní (anglicky *user interface*, zkratka UI) jako část počítače a jeho software, který mohou uživatelé vidět, slyšet nebo jinak vnímat či řídit. Dále se dá UI rozdělit na dvě části, vstup a výstup. Vstup je způsob jakým člověk komunikuje s počítačem například klávesnice nebo myš. Na rozdíl od toho je výstup komunikace počítače s uživatelem pomocí například počítačového monitoru. Dále v knize popisují vhodné uživatelské rozhraní jako kombinaci dobře navržených vstupních a výstupních mechanismů, které nejefektivněji uspokojí potřeby uživatele. Uživatelské rozhraní existuje ve dvou variantách: grafické nebo textové.

Článek [23] zase popisuje uživatelské rozhraní jako prostředek, který pomáhá uživateli komunikovat s rozhraním produktu pro služby, které uživatel požaduje. Dále zahrnuje prvky vizuálního designu včetně barev a typografie.

3.2.2 Uživatelská zkušenost

Článek [23] vysvětluje termín uživatelská zkušenost (anglicky *user experience*, zkratka UX) jako komunikaci prostřednictvím rozhraní, ze kterých uživatelé získávají zkušenosti s produktem a službami dané organizace. Dále vysvětlují, že je třeba provádět výzkumy za cílem zjištění pozitivních a negativních zkušeností od uživatelů na zlepšení UX.

3.2.3 Nástroje pro návrh uživatelských rozhraní

Článek [23] popisuje tři nástroje pro návrh uživatelských rozhraní.

1. **Figma:** je vektorový grafický editor a nástroj dostupný jak na internetu, tak v podobě aplikace. Pomocí Figmy je možné dělat design webových aplikací, mobilních aplikací, vektorové grafiky, atd. Figma je lehká na naučení a také podporuje kolaboraci v reálném čase. Pomocí mobilní aplikace je možné prohlížet prototypy rovnou na mobilním zařízení. Tato aplikace je zdarma, ale obsahuje také placené funkce pro vývojáře.

2. **Adobe XD:** je vektorový nástroj pro návrh UX 3.2.2 designu. V Adobe XD je možné vytvářet design pro webové i mobilní aplikace. Adobe XD podporuje sdílení souborů pomocí svého kreativního cloudu (*Creative Cloud*) mezi ostatními aplikacemi od Adobe.

3. **Sketch:** je stejně jako předchozí dvě aplikace také vektorový grafický editor, který vytváří UX a UI design. Sketch je dostupný pouze na macOS zařízeních.

3.2.4 Principy tvorby uživatelských rozhraní

Na základě analýzy článek [18] definuje několik základních atributů pro tvorbu použitelných uživatelských rozhraní.

1. **Vzhled:**

   - *Smyslový komfort* – u software se jedná hlavně o zrakové a sluchové vjemy. Vizuální komfort můžeme zajistit pomocí výběru vhodné kombinace barev, které zajišťují čitelnost. Je také důležité myslet na osoby s poruchou vnímání barev.
   - *Minimalismus* – je důležité se řídit slovním spojením „v jednoduchosti je krása“. Přílišná kombinace ornamentů a barev vede k nepohodlí při práci s uživatelským rozhraním.

- *Intuitivnost* – uživatelské rozhraní by měl zvládnout ovládat i člověk, který není zkušený a zaučený na dané platformě.
- *Estetika* – design nesmí být strohý. Musí obsahovat správnou kombinaci barev, písem a grafických prvků. Estetický design je předpokladem pro spokojené uživatele.
- *Struktura rozhraní* – správné uspořádání prvků rozhraní.

2. **Konzistence:**
   - *Standardizace* – je dobré používat zavedené normy (ISO, IEC, IEEE a další) nebo normy zavedené ve společnosti, pro kterou implementujeme uživatelské rozhraní. Tím se zajistí jednotnost uživatelských rozhraní napříč firmou a jejími produkty.
   - *Konvence z reálného světa* – využití objektů co lidé znají z reálného světa a jejich využití ve virtuálním světě.
   - *Mentální modely uživatelů* – pokud návrh odpovídá předsudkům a představám uživatelů, může být používání uživatelského rozhraní pro uživatele jednodušší a snazší.
3. **Zpětná vazba:**
   - *Doplňující informace* – je dobré zakomponovat do uživatelského rozhraní sekci pro rady, často kladené otázky, odkazy na nové funkce nebo dokumentaci.
   - *Vhodné popisky a průběh práce systému* – zobrazování vhodných zpráv pro uživatele na označení dalšího postupu. Je také důležité zobrazovat co se děje v systému.
4. **Efektivita:**
   - *Snadná práce* – Uživatel musí rychle a snadno dosáhnout svých specifických cílů.
   - *Pocit svobody* – Je vhodné umožnit uživatelům vypnout a znovu spustit rozhraní.
   - *Nadměrné užití kognitivních zdrojů* – Pokud je uživatelské rozhraní neintuitivní nebo požaduje nadměrnou pozornost na provádění základních úkolů, může to způsobit nadměrnou kognitivní zátěž.
   - *Různorodý přístup k dosažení cílů* – Je dobré implementovat ten samý úkol několika způsoby. Tím se mohou oslovit různé skupiny uživatelů například začátečníci, mírně pokročilí a další.
5. **Prevence chyb:** umožní uživatelům provádět stejný úkol několika způsoby, aby bylo možné při nalezení chyby použít cestu jinou.

3.3 Znovupoužitelné grafické komponenty

Stránka [9] popisuje znovupoužitelnost jako schopnost použít stejný zdroj vícekrát, více způsoby a ve více kontextech. Dále zmiňuje použití již existujících komponent pro vytvoření dalších komponent, což se stává velmi oblíbenou praxí. Mluví také o ušetření práce a času vývojářů při rozšiřování aplikace. Webové aplikace se stávají složitější a komplexnější. Znovupoužitelné komponenty udržují kratší doby vývoje těchto aplikací. Dále mluví

o interaktivnějším a přizpůsobivějším uživatelském rozhraní díky znovupoužitelným grafickým komponentám.

Podle článku [5] jsou znovupoužitelné komponenty předem připravené modulární prvky, které lze použít v různých částech softwarové aplikace, webových stránek nebo jakéhokoli jiného projektu. Také jsou tyto komponenty navrženy tak, aby plnily konkrétní funkci nebo poskytovaly určitý prvek uživatelského rozhraní (UI). Jsou vytvořeny tak, aby je bylo možné snadno integrovat do různých částí projektu, aniž by bylo nutné je pokaždé vytvářet znovu od začátku. Také tento článek definuje výhody znovupoužitelných grafických komponent.

1. **Efektivita:** eliminují nutnost začínat s každým novým projektem od nuly. To urychluje proces vývoje a umožňuje vývojářům soustředit se na zdokonalování a přizpůsobování prvků namísto jejich nového vymýšlení.

2. **Konzistence:** zajišťují jednotný vzhled v různých částech aplikace nebo v několika projektech, což zvyšuje uživatelský komfort a identitu značky.

3. **Inovace:** díky tomu, že vývojáři staví na stávajících prvcích designu, mohou efektivněji inovovat při zachování známého uživatelského prostředí.

4. **Úpravy a udržitelnost do budoucna:** s vývojem technologií se objevují nová zařízení a platformy. Při změnách v designu webové aplikace stačí pouze změnit komponentu na jednom místě.

Kapitola 4

Analýza

Tato kapitola se zaměřuje na analýzu stávajícího řešení grafické knihovny aplikace ACADA a její nedostatky v rámci požadovaného řešení. Základní požadavky na řešení byly stanoveny společností Logimic. Další důležité podmínky jsou stanoveny principy tvorby uživatelských rozhraní jako například jednotnost vzhledu, ale také principy pro tvorbu znovupoužitelných grafických komponent.

4.1 Požadavky na řešení

Od společnosti Logimic je to revize již existujícího grafické knihovny pro zlepšení uživatelské zkušenosti (*UX*). Na to navazující sjednocení grafického vzhledu komponent. Dále vytvoření nových komponent pro uživatelskou úpravu aplikace. Nové komponenty budou navrženy tak, aby mohly být použity v různých částech aplikace a pro různé účely. To povede k efektivnějšímu a kompaktnějšímu kódu a usnadní implementaci nových funkcí v budoucnu. Zároveň je také důležité se řídit cílovými skupinami uživatelů již existující grafické knihovny pro zlepšení jejich zkušenosti.

4.1.1 Cílové skupiny

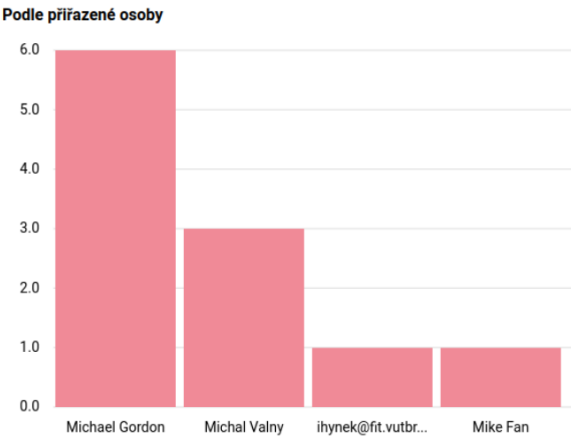
Při analýze je zapotřebí se podívat na produkt různými pohledy pro pochopení požadavků na grafické komponenty. První cílová skupina jsou vývojáři této platformy. Vývoj v nepříznivém prostředí je velmi náročná činnost. Z tohoto důvodu je důležité zamyslet se nad prací vývojářů. Důležitá část implementace sestává z vytvoření uživatelského rozhraní. Při individuálním řešení uživatelského rozhraní na každé stránce aplikace zvlášť, se narušuje jednotnost celé aplikace. Také se tím prodlužuje čas strávený na implementaci. Potřeba vývojáře je zjednodušení rozšířitelnosti aplikace prostřednictvím sady znovupoužitelných grafických komponent. Když programátor vytvoří novou stránku, tak by mělo stačit vzít již existující komponenty a poskládat je dle potřeby na stránku. Tato možnost sníží čas strávený na tvorbě uživatelského rozhraní. Také je důležité se zamyslet nad správnou atomizací daných komponent a případně nad jejich vstupními rozhraními.

Další cílovou skupinou jsou uživatelé aplikace. Tito lidé potřebují přehledné zobrazení základních informací na řídícím panelu 2.3.1 pro rychlou kontrolu stavu jednotlivých zařízení. Také chtějí aplikaci, u které nebudou muset nadměrně využívat mozkovou kapacitu. Uživatel by měl aplikaci zvládnout používat bez pomoci. Z tohoto důvodu je potřeba mít komponenty, které jsou srozumitelné a jednoduché na ovládání. Další metoda ke zlepšení uživatelského zážitku je konfigurovatelnost hlavního řídícího panelu.

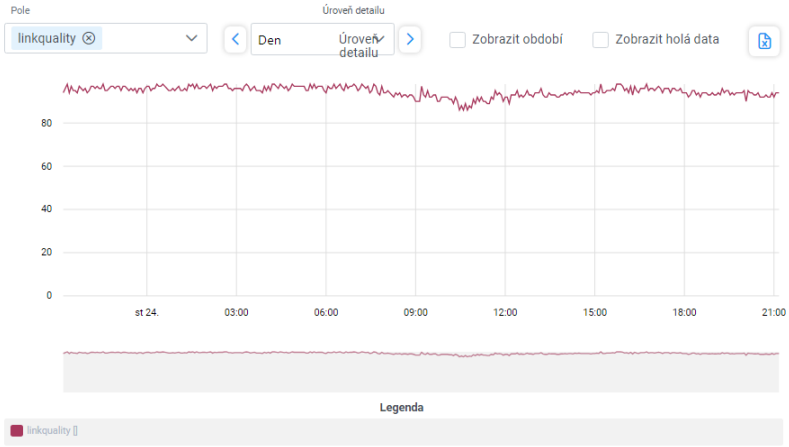
4.2 Současný stav grafické knihovny

Aplikace ACADA již disponuje základní grafickou knihovnou obsahující grafické komponenty od firmy Logimic. Tyto grafické komponenty jsou z větší části založené na grafické knihovně PrimeNG[1]. Dále se grafické komponenty dělí do dvou základních skupin: komponenty pro vizualizaci dat a komponenty pro ovládání. Aby bylo možné zobrazit všechny typy dat, je zapotřebí mít různé typy komponent pro vizualizaci dat. Jako první jsem zkoumal analytické komponenty. Jsou to komponenty, které analyzují data z minulosti a zobrazují je přehledně uživateli. Do této kategorie spadají hlavně grafy. Ty se v aplikaci nacházejí tři: sloupcový, čárový a odrážkový graf, jak je možné vidět na obrázku 4.1. Po konzultaci požadavků firmy bylo usouzeno, že není zatím potřebné grafy předělávat.

(a) Odrážkový graf



(b) Sloupcový graf



(c) Čárový graf

Obrázek 4.1: Zobrazení aktuálního stavu grafů v aplikaci ACADA

Dále se používají komponenty operační. Tyto komponenty vizualizují data v reálném čase. Používají se na zobrazení aktuálního stavu zařízení.

- **Kruhový indikátor** (*circle indicator*): hlavní ukazatel stavu již přidaných zařízení do cloudové platformy. Obsahuje také KPI jako tečku pod kruhem a jejich stav se

[1] https://primeng.org

dá určit podle barvy této tečky, jak je možné vidět na obrázku 4.8. Dále je tečka popsána jménem daného KPI ukazatele. Adresář stále obsahuje dvě předchozí verze kruhových indikátorů, které nahradil tento nový. Problém této komponenty je, že není responzivní, protože má pevně nastavenou velikost.

V aplikaci je zapotřebí se pohybovat mezi stránkami. To zajišťují komponenty obsažené ve většině webů jako například navigace. Grafický adresář obsahuje navigační lišty pro zobrazení na velkém zařízení, ale i na mobilním zařízení. Také obsahuje drobečkovou navigaci, která umožní pohyb zpět.

- **Navigační lišta** (*navbar*): skládá se z dalších dvou komponent, které se následně kombinují do jedné navigační lišty pro mobil i jiné zařízení. Jak je vidět na obrázku 4.8, tak navigační lišta je barevně konzistentní s aplikací a je podle požadavků firmy.
  - **Responzivní navigační lišta** (*responsive navbar*): stará se o vzhled navigační lišty na různých zařízeních. Na telefonu se zobrazuje přeš celou obrazovku. Na počítači zabírá pouze malé místo na levé straně obrazovky. Při najetí myší na lištu se velikost zvětší.
  - **Navigační hamburger** (*navbar hamburger*): zobrazuje se pouze na menších zařízeních pro otevření menu přes celou obrazovku. Na obrázku 4.2b je vidět zobrazený hamburger.
- **Drobečková navigace** (*breadcrumbs*): grafická komponenta sloužící pro navigování zpět na předchozí otevřené stránky. Používá se v celé aplikaci. Na obrázku 4.2a je možné vidět barevně konzistentní vzhled se zbytkem aplikace.

(a) Drobečková navigace

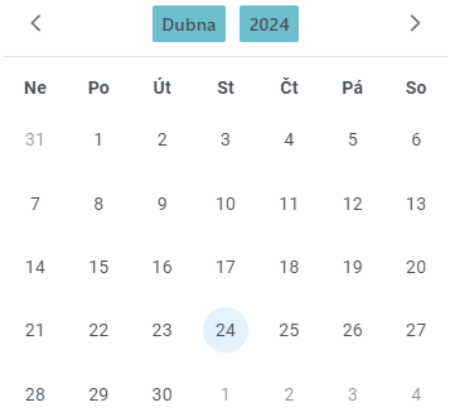
(b) Mobilní zobrazení navigačního hamburgeru

Obrázek 4.2: Navigační komponenty

Dále se tu nachází komponenty potřebné pro získávání vstupů od uživatelů. **Kalendář** (*calendar*) se používá pro výběr určitého času, také je možné vybrat interval (datum a čas). Používá se například pro výběr intervalu pro statistiky zařízení. Není vzhledově konzistentní s aplikací, jak je zobrazeno na obrázku 4.3.

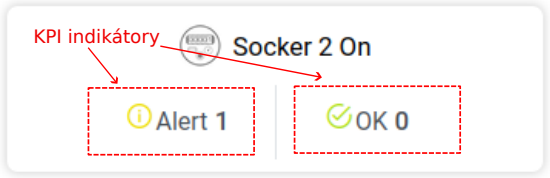
4.2.1 Problémové komponenty

V této sekci jsou rozebrány komponenty, které mají více nekonzistencí nebo nesplňují požadavky firmy. Jako první mezi ně patří indikátor KPI (*kpi indicator*). Jedná se o komponentu zobrazující ikonu, název a hodnotu určitých upozornění. V aplikaci se používá dvojice této komponenty na kartičce, která neexistuje jako samostatná komponenta, jak je vidět na obrázku 4.4a. Při implementaci podobného zobrazení na jiné obrazovce bude potřeba zase vytvořit tuto kartičku. Stará verze této kartičky s názvem *ifield* obsahovala i nadpis a obrázek KPI, jak je vidět na obrázku 4.4b. V adresáři se také nachází komponenta čtveřice



Obrázek 4.3: Nekonzistentní vzhled komponenty pro kalendář se zbytkem aplikace.

(*foursome*). Tato komponenta byla také využívána pro podobné účely jako *ifield* jen s tím rozdílem, že obsahuje čtyři políčka pro indikaci. Takže vzhledově nezapadá do aplikace, jak můžeme vidět na obrázku 4.5.



(a) Kartička pro zobrazení KPI a jeho upozornění

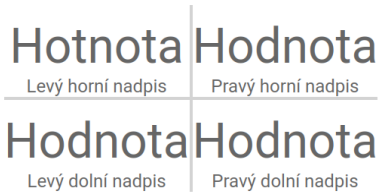
(b) Stará verze zobrazování KPI

Obrázek 4.4: Porovnání staré verze zobrazování KPI a nové verze

Při dalším zkoumání jsem nalezl výstražnou kartu (*alert card*), která zobrazuje hodnotu nebo výběr z hodnot. Karta má tři varianty: zobrazení hodnoty, rozbalovací seznam (anglicky dropdown) na výběr možnosti nebo vlastní typ. Dropdown variantu a zobrazení hodnoty je možné vidět na obrázku 4.8. V dalších částech aplikace existují kartičky se stejným vzhledem, ale tyto kartičky jsou vytvořeny zase individuálně. Porovnání kartiček je na obrázku 4.6.

Dále jsem zjistil, že se nepoužívá jednotný styl pro tlačítka. Komponenta pro tlačítko (*button*) se nevyužívá a každé tlačítko vypadá jinak. Také jsem prohlížel jednotlivé tabulky v aplikaci a použitá tlačítka v nich. Jak je možné vidět na obrázku 4.7, každé tlačítko vedle vyhledávacího řádku má jinou barevnou kombinaci.

Poslední důležitou částí, které jsem se věnoval, je hlavní řídící panel (*dashboard*). Řídicí panel je sice sestaven z již hotových komponent, jak je vidět na obrázku 4.8, ale chybí zde jedná zásadní funkcionalita požadována firmou Logimic. Uživatel by si rád rozmístil různé komponenty sám svým způsobem. Aby toto bylo možné je zapotřebí mít na hlavní stránce vysázené komponenty v mřížce. Taková komponenta mřížky (*grid div*) již v aplikaci existuje,



Obrázek 4.5: Ukázka vzhledu komponenty čtveřice

(a) Výstražná kartička

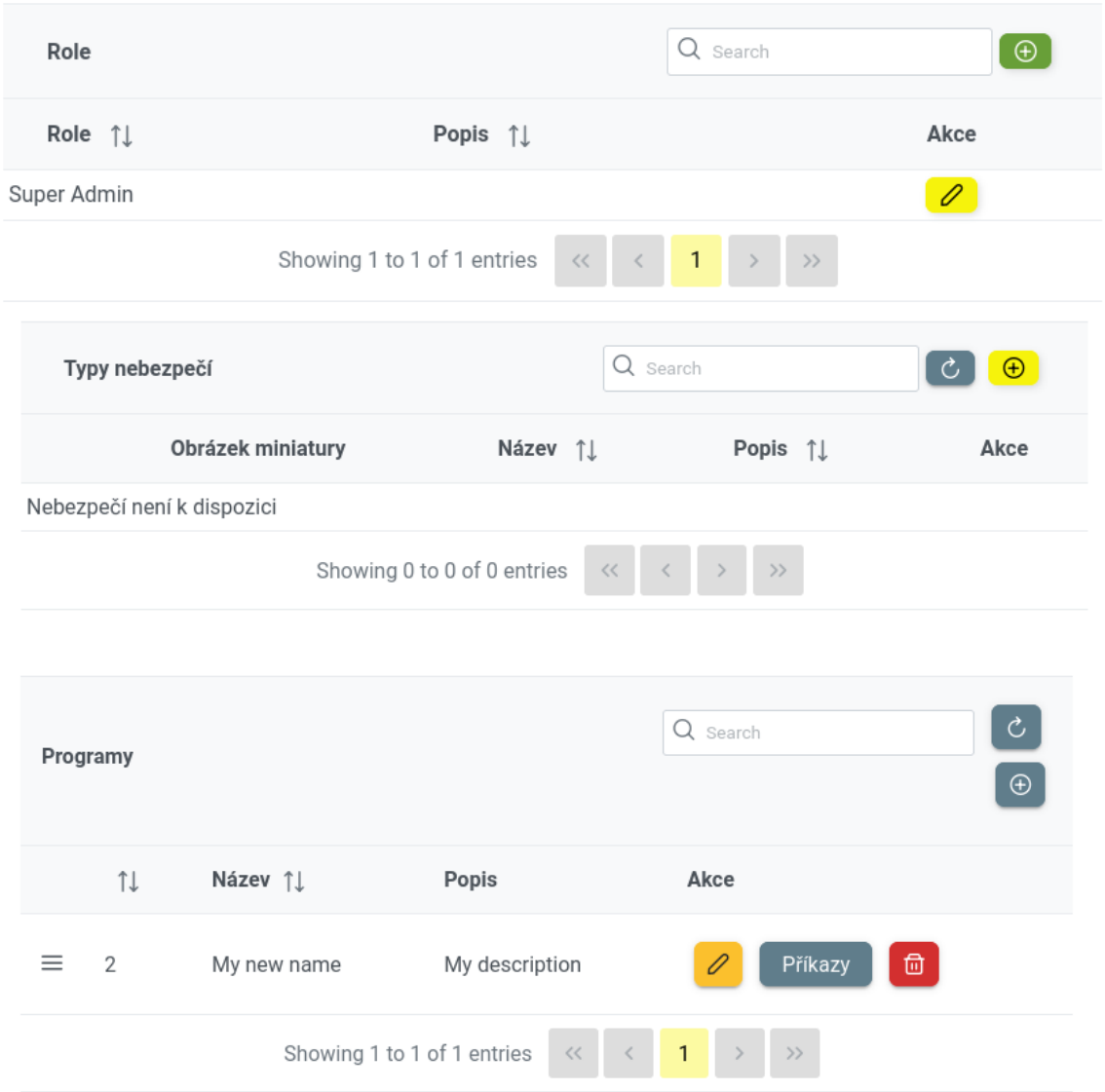
(b) Kartička tvořená individuálně

Obrázek 4.6: Porovnání dvou kartiček dostupných v aplikaci ACADA

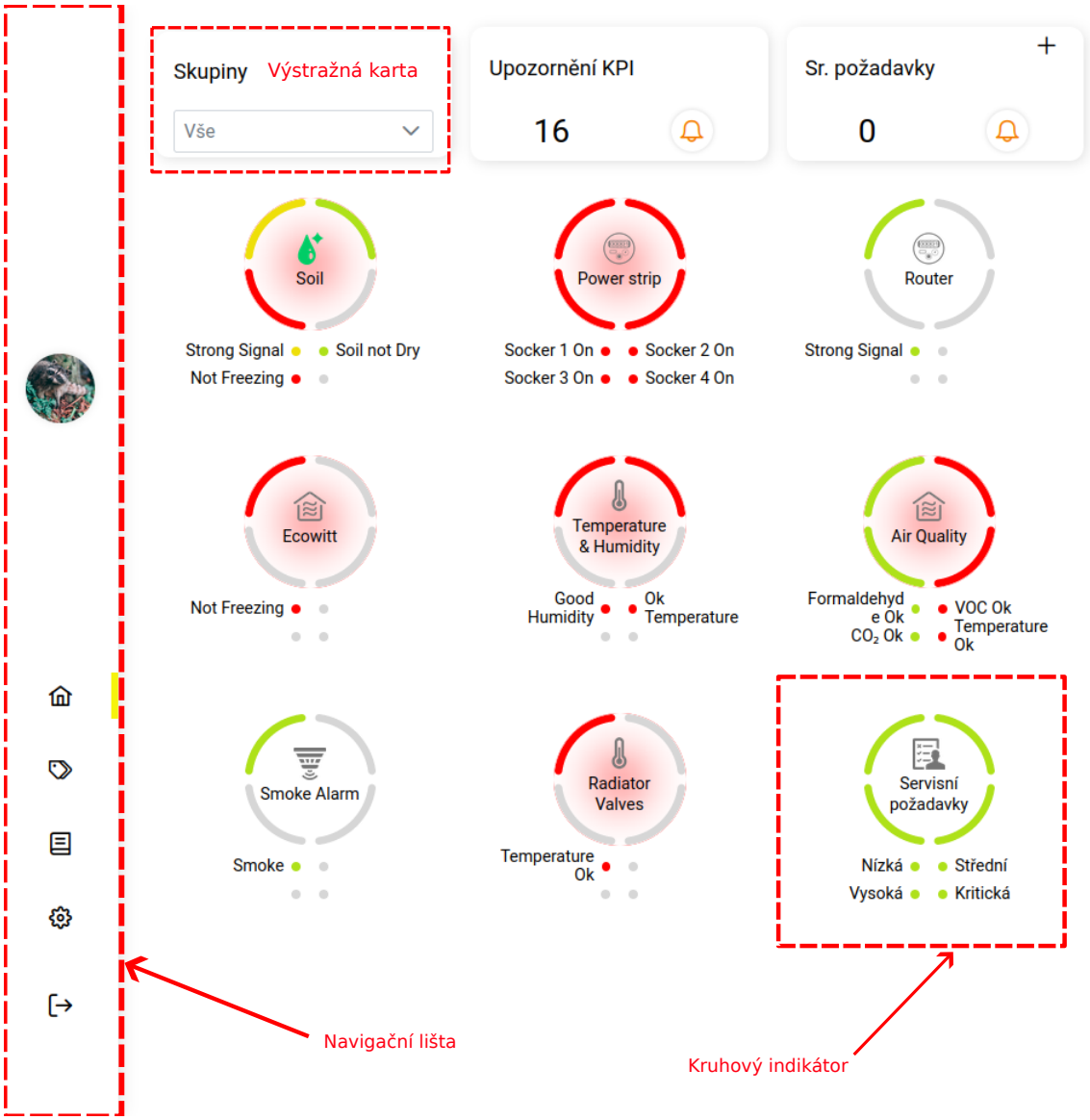
ale jedná se pouze o statické rozložení. Dále by měla být mřížka dynamická pro možnost vložení jakékoliv komponenty. Také by měla být responzivní a měnit počet sloupečků podle velikosti obrazovky.

4.3 Závěr analýzy

V důsledku analýzy grafického adresáře aplikace ACADA jsem zjistil, že vzhled v některých částech aplikace není konzistentní. Některé stránky v aplikaci mají své individuální řešení, které není sdíleno v grafickém adresáři a šlo by použít na více místech. Je zapotřebí vytvoření některých komponent pro další použití. Dále je dobré snížit počet různých karet, sjednotit je a použít je v aplikaci. Zjednoduší se tím práce vývojářům při hledání správné komponenty. V rámci požadavků od uživatelů je dobré mít komponenty jednoduché a přehledné například barevné kombinace tlačítek jsou v tabulkách na každé stránce jiné. Také je dobré se zaměřit na konfigurovatelnost řídícího panelu pro lepší uživatelskou zkušenost.



Obrázek 4.7: Obrázek ukazuje nekonzistenci tlačítek napříč aplikací. Ta se například nachází u tabulek.



Obrázek 4.8: Starý dashboard aplikace ACADA obsahující na levé straně navigační lištu, která se po najetí myší rozvine a zobrazí názvy dostupných možností. Dále výstražné kartičky na vrchní straně dashboardu. Tyto kartičky se nacházejí v horní části obrazovky a informují uživatele o důležitých událostech, jako jsou upozornění na KPI a servisní požadavky. Kruhové indikátory se nacházejí v centrální části obrazovky a vizuálně reprezentují stav jednotlivých skupin zařízení. Indikátory mění barvu v závislosti na stavu dané skupiny (například zelená pro „v pořádku“, červená pro „problém“).

Kapitola 5

Návrh

Hlavním požadavkem Logimic bylo vyvinutí víceúčelových komponent. Tyto komponenty by měly zahrnovat škálu funkcí a mohly by být použity v různých kontextech aplikace. Důraz byl kladen na snadnou přizpůsobitelnost pro uživatele, která by jim umožnila upravovat rozložení aplikace dle specifických potřeb. Zároveň bylo nezbytné zachovat responzivní design, aby se komponenty správně zobrazovaly a fungovaly na různých zařízeních a platformách. Kromě výše uvedených požadavků se zaměříme i na dosažení maximální jednotnosti vzhledu celé aplikace a zajištění optimální použitelnosti pro uživatele. Jednotný design zajistí konzistentní uživatelský dojem a usnadní orientaci v aplikaci.

5.1 Nově vytvořené komponenty

Aby byla knihovna grafických komponent komplexní, bylo zapotřebí vytvořit základní stavební komponenty pro rozvržení aplikace. Z těchto komponent je poté možné jednoduše sestavit jakoukoliv stránku v aplikaci.

5.1.1 Patička (anglicky footer)

Z důvodu absence jakékoliv základní komponenty pro patičku jsem navrhl jednoduchý vzhled patičky, jak je možné vidět na obrázku 5.1. **Vstupy** pro komponentu *footer* jsou:

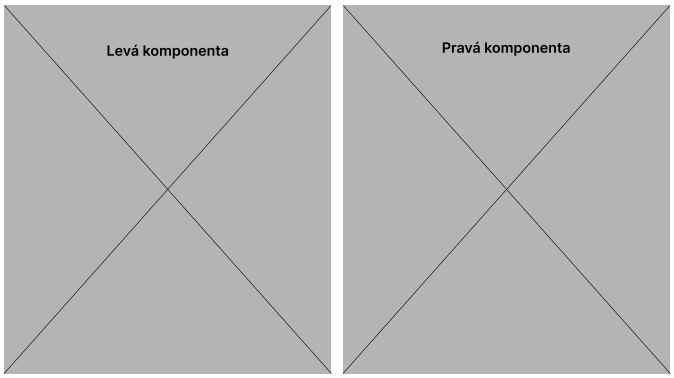
- *Pole akcí patičky* – obsahuje jednotlivé akce a jejich objekt se skládá z *názvu* a *odkazu*, na který je po stisku uživatel přesměrován. Je možné vložit jakýkoliv počet odkazů. Použít se dají například pro odkazy na dokumentaci aplikace nebo na oficiální stránky firmy Logimic.
- *Text patičky* – se nachází pod výše uvedenými odkazy. Jedná se o text, který může sloužit pro jakékoliv sdělení pro uživatele.
- *Pravý horní text* – může sloužit například jako ukazatel názvu aplikace. Nachází se v pravé části patičky a je výrazný.
- *Pravý dolní text* – naopak není tak výrazný ani velký. Může být použit pro méně důležité informace například verzi aplikace.

Obrázek 5.1: Návrh pro základní patičku stránky obsahující tři možné umístění pro text. Je zde možné vypsat různý počet odkazů.

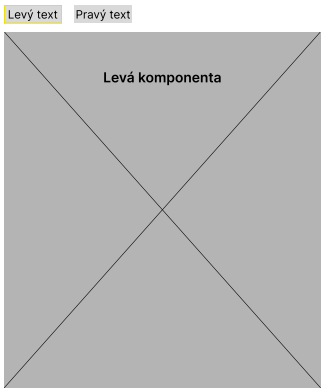
5.1.2 Responzivní rozložení (anglicky responsive layout)

Pro lepší organizaci stránek jsem navrhl komponentu pro rozložení. Bude díky tomu možné umístit jakékoliv dvě komponenty vedle sebe a nebude nutné řešit rozložení na každé stránce zvlášť. Dvousloupcové rozložení bude plně responzivní a automaticky se přizpůsobí šířce displeje. V závislosti na šířce displeje se zobrazí komponenty vedle sebe nebo se zobrazí záložky pro přepínání mezi nimi. **Vstupy** pro tuto komponentu obsahují:

- *Levá a pravá komponenta* – bude obsahovat objekt celé levé nebo pravé komponenty pro jejich vykreslení. To znamená i jejich vstupy a výstupy.
- *Levý a pravý text* – se bude zobrazovat na mobilním zařízení jako popis jednotlivých záložek pro přepínání.
- *Výchozí záložka* – označuje, která z dvou komponent se má při načtení zobrazit jako první. Volitelný vstup, který ve výchozím stavu zobrazuje první záložku.



(a) Varianta pro velké zobrazení



(b) Varianta pro mobilní zobrazení

Obrázek 5.2: Nová komponenta *responzivní rozložení*, která umožňuje zobrazit dvě různé komponenty vedle sebe na větších displejích. Na menších zařízeních, jako jsou mobilní telefony, se tato komponenta automaticky přepne do režimu s tlačítky pro přepínání mezi jednotlivými komponenty. Toto responzivní chování zajišťuje optimální zobrazení obsahu pro všechny typy zařízení.

5.1.3 Konfigurovatelná mřížka (anglicky grid)

Tato mřížka by měla umožnit uživatelům přemisťování jednotlivých komponent na jiná místa. To povede k větší personalizaci rozhraní. Komponenty v mřížce budou moci uživatelé roztahovat do více políček, čímž si zvětší jejich zobrazovací plochu. To bude užitečné

pro komponenty, které obsahují více informací nebo vyžadují detailnější zobrazení. Také bude možné komponentu odstranit. Při takovýchto úpravách je zapotřebí vidět ohraničení jednotlivých polí. Proto bude mřížka mít dvě zobrazení: editační mód 5.4 a normální mód. Z důvodu responzivního designu bude mřížka měnit počet sloupečků na základě velikosti obrazovky. Na mobilních zařízeních s menším displejem se hlavní panel zobrazí jako seznam komponent, čímž se zajistí optimální použitelnost i na menších zobrazeních. **Vstupy** pro komponentu *grid* jsou:

- *Pole komponent* – definuje jaké komponenty se mají v *gridu* vypsat a na jakých místech. Objekt pro jednu komponentou je možné vidět v kódu 5.1. Tento objekt obsahuje její typ. Dále jsou důležité její vstupy. Poté je v tomto objektu důležité definovat místo zobrazení (`poziceGrid`) a jeho velikost (`velikostGrid`).
- *Povolení odstranění komponenty* – bude použit v případě, že vývojář plánuje zavést podporu pro odstranění. Ve výchozím stavu není možné komponenty odstranit.

```
komponentaProGrid: {
    komponentaProVykresleni,
    poziceGrid: {
        poziceX,
        poziceY,
    }
    velikostGrid: {
        velikostX,
        velikostY,
    }
}
```

Výpis 5.1: Návrh objektu pro vstupní definici komponenty. Také definuje její pozici a velikost na obou osách.

5.1.4 Widget

Pro správnou funkčnost konfigurovatelné mřížky je důležité udělat dynamické vykreslení jakékoliv komponenty. Následně bude stačit jen předat typ komponenty, který se má využít. Dále seznam vstupních parametrů, které ovlivňují vzhled a chování komponenty. Také by měla podporovat výstupní parametry pro propagaci dat opačným směrem. **Vstupy** do komponenty *widget*:

- *Typ komponenty* – určuje, která komponenta se má vykreslit. Musí podporovat jakoukoliv komponentu z grafické knihovny.
- *Vstupní parametry* – určené podle typu komponenty. Jedná se o seznam parametrů, které ovlivňují vzhled a chování komponenty. Například barva tlačítka, data v tabulce, typ grafu.
- *Výstupní parametry* – také určuje typ komponenty. Tato komponenta musí správně propagovat výstupy do nadřazených komponent. Například stisknutí tlačítka, výběr řádku v tabulce, změna hodnoty v grafu.

5.1.5 Karta indikátorů KPI (anglicky KPI indicators card)

Tato nová komponenta by měla využívat ke svému fungování již existující indikátory KPI. Tyto indikátory obsahují pouze obrázek, hodnotu a nadpis hodnoty. Z důvodu používání skupin indikátorů jsem navrhl kartu, která seskupí tyto indikátory. Na základě staré komponenty *čtveřice* (viz obr. 4.5) jsem rozšířil funkčnost o zobrazování více jak dvou indikátorů KPI. **Vstupy** do komponenty *karta indikátorů KPI*:

- *Titulek karty* – se vykreslí na vrchní straně karty. Tento titulek označuje skupinu indikátorů.
- *Obrázek karty* – se nachází hned vedle názvu karty. Jedná se o vizuální označení skupiny.
- *Pole indikátorů KPI* – obsahuje objekty, které popisují indikátory KPI. Tento objekt je možné vidět v kódu 5.2. Na základě délky uvedeného pole se komponenta vykreslí jako jednořádková nebo dvouřádková, jak je možné vidět na obrázku 5.3.

```
indikatorKPI: {
    hodnotaIndikatoru,
    obrazekIndikatoru,
    barvaObrazku,
    popisHodnoty,
}
```

Výpis 5.2: Jedná se o navrhnutý objekt pro definování indikátoru KPI.

(a) Jednořádková varianta

(b) Víceřádková varianta

Obrázek 5.3: Nová KPI karta s indikátory

5.2 Přepracování existujících komponent

Při návrhu konfigurovatelného panelu s proměnlivou velikostí polí jsem objevil problematiku neměnných komponent. V rámci předchozí knihovny byly navrženy tak, aby se jejich velikost a vzhled neměnily v závislosti na rozlišení displeje. Pro řešení tohoto problému bylo nutné navrhnout responzivní přístup, který zajistí optimální zobrazení komponenty i na menších zařízeních. Mezi tyto komponenty patří *kruhový indikátor* 5.4, *výstražná kartička* 5.4 a dále nová varianta *karty pro indikátory KPI*.

Obrázek 5.4: Režim úprav řídícího panelu. Tento režim zahrnuje zobrazení ohraničení pro případné přesunutí jednotlivých karet. Na pravé straně se nachází tlačítko pro uložení konfigurace a přidání nové komponenty. Také jsou zde navrženy responzivní varianty *výstražné kartička* a *kruhového indikátoru*.

Kapitola 6

Implementace

Tato kapitola popisuje použité technologie a důvod jejich výběru pro danou implementaci. Dále je popsána implementace jednotlivých grafických komponent a jejich vstupní rozhraní.

6.1 Použité technologie

Firma Logimic vyvíjí aplikaci ACADA s využitím frontendového frameworku Angular 6.1.1. Z důvodu použití nově vytvořených komponent v aplikaci ACADA, bylo nutné využít programovací jazyk, ve kterém je napsaná.

6.1.1 Angular

Oficiální dokumentace definuje Angular [1] jako vývojovou platformu založenou na jazyku TypeScript 6.1.2. Angular je framework, jehož základním stavebním kamenem je komponenta, pro tvorbu rozšiřitelných webových aplikací. Dále obsahuje sbírku knihoven pro širokou škálu funkcí například směrování, správy formulářů a komunikace klient-server.

Jak bylo zmíněno, základní stavební kámen Angularu je *komponenta*. Ta se skládá ze šablony psané v *HTML*[1], kaskádových stylů (*CSS*[2]) a samotné logiky v jazyce *TypeScript*. Každá komponenta musí obsahovat a exportovat minimálně jednu *třídu* (anglicky *class*). Jelikož zakládá Angular na třídách, tak se jedná o objektově orientované programování (*OOP*).

Kromě komponent se v Angularu nachází i další klíčové koncepty, které usnadňují vývoj rozšířitelných aplikací. Patří mezi ně služby (*services*), direktivy (*directives*), potrubí (*pipes*) a moduly (*modules*). Services mohou sloužit pro sdílení logiky do více komponent naráz. Directivy se používají v šablonách, například `*ngIf` pro přidání podmínky na HTML tag nebo `*ngFor` pro dynamické renderování stejného kódu s jinými daty. Pipes se používají pro transformaci surových dat na data pro vykreslení do šablony. Moduly sdružují komponenty, které patří k sobě a tím vytvářejí logické celky.

6.1.2 TypeScript a JavaScript

Kniha [8] popisuje JavaScript jako odlehčený interpretovaný programovací jazyk s možností objektově orientovaného programování. Původně byl používán pro psaní skriptů na serverech. Pro použití na webu bylo nutné přidat objekty reprezentující okno webového

[1] Hypertextový značkovací jazyk, anglicky Hypertext Markup Language
[2] anglicky Cascading Style Sheets

prohlížeče a jeho obsah. Tato verze JavaScriptu umožňuje zahrnout spustitelný obsah do webových stránek. To znamená, že je možné dynamicky vytvářet HTML obsah. Také je to umožněno faktem, že jsou do webových prohlížečů zabudovány interprety pro scripty. Z jazyku JavaScript vychází TypeScript[3], který je vyvíjen firmou Microsoft. Základem je JavaScript doplněný o typy a jejich silnou kontrolu. Tyto typové kontroly zabraňují neočekávanému chování kódu a snižují šanci programátorské chyby. Jazyk TypeScript je prvně přeložen do jazyku JavaScript a až poté interpretován.

6.2 Detailní implementace

Při implementaci jsem začal s komponenty, které bylo potřeba vytvořit. Mezi tyto komponenty patří: grid, widget pro vykreslení jakékoliv komponenty, responzivní rozložení, kartička pro indikátor KPI a patička. Dále se pro některé komponenty implementovali globální styly z důvodu pouze vizuálních změn a použití existujících komponent z knihovny PrimeNG.

6.2.1 Konfigurovatelná mřížka

Z důvodu potřeb firmy Logimic bylo nutné vytvořit konfigurovatelnou mřížku (dále jen grid), ve které je možnost upravit si vlastní rozložení jednotlivých komponent. Základem gridu je knihovna Gridster2[4]. Ta podporuje přesouvání a zvětšování jednotlivých položek. V šabloně se na základě knihovny používá komponenta `gridster`, ten požaduje jako vstup nastavení `options`, jak je vidět na kódu 6.1.

```
<gridster
  [options]="options">
  <gridster-item
    [item]="item.gridPosition"
    *ngFor="
      let item of components
      ">
        <!--dynamic component-->
  </gridster-item>
</gridster>
```

Výpis 6.1: Základní kostra gridu v HTML šabloně.

```
options: {
    gridType: 'scrollVertical',
    maxCols: 10,
    setGridSize: true,
    displayGrid: 'none',
    resizable: { enabled: false },
    draggable: { enabled: false },
    mobileBreakpoint: 768,
    }
```

Výpis 6.2: Vstupní objekt použitý v této bakalářské práci pro nastavení gridu

Pro potřeby této bakalářské práce jsem použil jen některá nastavení dostupná pro komponentu `gridster`. Vstupní objekt s jeho parametry je možné vidět v kódu 6.2. Tento objekt obsahuje:

- **Typ gridu** (`gridType`): obsahuje možnosti zobrazení gridu na stránce a jeho chování. Tyto možnosti jsou:
  - `fit` – se snaží všechny komponenty vměstnat na obrazovku bez použití posuvníku.

[3] https://www.typescriptlang.org
[4] https://github.com/tiberiuzuld/angular-gridster2

  - `fixed` – musí se nastavit pevná výška a šířka jednoho okénka.
  - `fixedHorizontal a fixedVertical` – určují pevnou velikost pouze na jedné ose. Další osa se řídí pomocí typu `fit`.
  - `scrollHorizontal a scrollVertical` – nastaví na jedné ose možnost posunutí pomocí posuvníku. Ta druhá se řídí zase pomocí typu `fit`.

  Pro využití gridu na hlavním panelu byla vybrána možnost `scrollVertical` z důvodu nežádoucího posouvání na ose horizontální, ale na ose vertikální není omezující.

- **Maximální počet sloupečků** (`maxCols`): omezuje sloupečky, které je možné mít na obrazovce. Tato proměnná se mění na základě velikosti obrazovky pomocí interní logiky konfigurovatelného gridu 6.2.1. Výchozí hodnota v rámci bakalářské práce byla nastavena na číslo 10.

- **Automatické nastavení polí** (`setGridSize`): přizpůsobí velikost jednotlivých polí nejmenší komponentě, která je vykreslená v gridu. Pro použití této bakalářské práce je tento parametr povolen.

- **Zobrazení gridu** (`displayGrid`): nastavení vykreslení ohraničení pro jednotlivá pole. Obsahuje tři možnosti zobrazení:

  - `always` – označuje neustálé zobrazení ohraničení.
  - `onDrag&Resize` – zobrazí ohraničení pouze pokud uživatel interaguje s velikostí položek nebo jejich pozicí.
  - `none` – nezobrazuje žádné ohraničení.

  Výchozí nastavená hodnota je `none` z důvodu nutnosti přepnutí se do editačního módu prezentovaného v návrhu 5.4. Poté se zobrazí ohraničení pomocí hodnoty `always`.

- **Změna velikosti a přemisťování** (`resizable, draggable`): obsahují objekt pro určení jakým způsobem mají tyto akce probíhat. Důležitým parametrem tohoto objektu je `enabled`, pomocí kterého se tato akce povoluje. Ve výchozím stavu jsou tyto akce zakázané. Grid musí být v editačním módu, jak bylo zmíněno v rámci návrhu.

- **Mobilní breakpoint** (`mobileBreakpoint`): požaduje jako vstup číslo. Toto číslo definuje hranici šířky. Když se obrazovka zmenší pod tuto hranici, tak se změní rozložení gridu pouze na jediný sloupec. Tento sloupec nebude možné upravovat a jedná se pouze o výpis.

Další důležitou součástí gridu jsou jednotlivé položky, které je možné rozmístit. Tyto položky jsou obalené v komponentě `gridster-item`. Do tohoto tagu je potřebné vložit vstupní objekt 6.3 obsahující `x` pro pozici na ose x, `y` pro osu y. Dále obsahuje `rows` pro počet řádků, které má komponenta zabírat a `cols` pro sloupce.

```
gridPosition: {
    x: number,
    y: number,
    rows: number,
    cols: number;
}
```

Výpis 6.3: Objekt vkládaný do `gridster-item` z knihovny Gridster2, pro určení pozice a velikosti v gridu.

Vstupy a výstupy

Pro dynamické vykreslování je důležité mít dobré vstupní pole s objektem, který dostatečně popíše komponentu pro její pozici a vykreslení. Základní objekt pro komponentu byl definován již v návrhu 5.1, ale na základě konzultací s Markem Konečným [13], jehož bakalářská práce na tuto práci navazuje, byl vytvořen nový objekt typu `LgmcGGridInput` 6.4. Tento objekt obsahuje:

- **Vstupní komponentu** (`component`): jedná se o položku typu `LgmcGWidgetInput`. Tento nový objekt využívá další nová komponenta `widget` 6.2.2. Obsahuje typ komponenty a její vstupy.
- **Pozice v gridu** (`gridPosition`): obsahuje více parametrů, které komplexně popisují pozici a velikost komponent. Tyto parametry jsou:
  - `order` – udává základní pořadí komponent, ve kterém se mají vykreslit do připraveného gridu. Pokud není nastavené individuální rozložení, použije se vždy pořadí.
  - `defaultSize` – popisuje základní velikost komponenty. To znamená, kolik zabere sloupců (`cols`) a kolik řádků (`rows`).
  - `largeScreenPosition` a `mediumScreenPosition` – označují specifické rozložení komponent (`coordinates`), ale také jejich velikost (`size`). Hranice pro velké obrazovky je šířka větší jak 1440px. Pro střední obrazovky se jedná o šířku větší jak 768px a menší nebo rovno 1440px.

Mezi další vstupy patří titulek nad gridem (`title`) a povolení pro odstraňování jednotlivých položek (`enableDeleteItems`). Ve výchozím nastavení není možné položky mazat.

```
component: LgmcGWidgetInput;
gridPosition: {
  order: number,
  defaultSize: {
    rows: number,
    cols: number,
  },
  largeScreenPosition: devicePosition,
  mediumScreenPosition: devicePosition,
}
```

Výpis 6.4: Vstupní objekt obsahující komponentu a její pozici pro vstup do gridu. Používá objekt pro pozici `devicePosition` 6.5.

```
devicePosition: {
    coordinates: {
        x: number;
        y: number;
      },
      size: {
        rows: number;
        cols: number;
      }
}
```

Výpis 6.5: Objekt používaný v implementaci bakalářské práce pro určení velikosti a pozice.

Interní implementace

Jako interní pomocnou proměnou pro snadnější vykreslování komponent a případné úpravy jejich pozice jsem si vytvořil pole `componentsWithPositions` s objekty typu 6.6. První problém byl, že použitá knihovna Gridster2 nepodporuje responzivní změnu sloupečků při zmenšení displeje. Proto bylo nutné vytvořit vlastní funkci `resizeGrid(widthOfScreen: number)`, která při změně šířky displeje změní počet sloupečků. Následně si uloží tuto velikost (*small*, *medium*, *large*) do proměnné `sizeOfScreen` pro využití v implementaci.

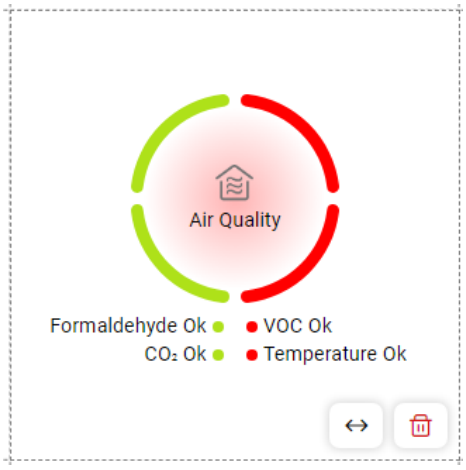
```
DefaultComponentWithGridsterPosition: {
  componentWithDefaultPosition: LgmcGGridInput;
  gridItemPosition: GridsterItem;
}
```

Výpis 6.6: Za účelem zjednodušení manipulace s daty komponent byl zaveden pomocný objekt. Ten zahrnuje základní objekt pro vstupní komponentu do gridu a dále obsahuje pozici, která je typu 6.3 a slouží pro vložení do komponenty `gridster-item`.

Další důležitou funkcionalitou je upřednostnění komponent, které mají nastavenou pozici před těmi bez nastavené pozice. Proto jsem vytvořil funkci:

- **`calculateOrderOfComponents`**: zabezpečuje použití správného rozložení a na vstup je přivedena proměnná `sizeOfScreen`. Tato funkce nejprve seřadí pole objektů typu `LgmcGGridInput` 6.4 podle položky `order`. Dále na základě `sizeOfScreen` vybere příslušné rozmístění. Pokud se jedná o mobilní zařízení nebo není pro danou komponentu na vybrané velikosti obrazovky nastavena pozice nastaví se x=0, y=0 a aplikuje se výchozí velikost komponenty (`defaultSize`). Umístění komponenty bude řízeno gridem a automaticky se vybere nejvhodnější volný prostor. Tyto údaje jsou poté uloženy v pomocném poli `componentsWithPositions`, které je předáno do šablony.

Také bylo zapotřebí implementovat funkce, které při zavolání zapnou nebo vypnout editaci. Pro přesunutí se do režimu úprav je zapotřebí zavolat funkci `startEditionGrid()`.



Obrázek 6.1: Položka gridu v editačním módu

Implementovaná funkce zobrazí grid s ohraničením a tlačítky pro interakci s jednotlivými položkami, jak je vidět na obrázku 6.1. Uživatelé tak mohou pomocí tlačítka se šipkami posouvat položky v gridu a tlačítkem pro odstranění je možné položky z gridu odebrat. Odebrání se zobrazí pouze v případě, že bylo na vstup `enableDeleteItems` vloženo `true`. Pro ukončení režimu editace gridu a uložení provedených změn slouží funkce `stopEditionGrid(saveChanges: boolean)`. Pokud má být editace ukončena bez uložení změn, je nutné funkci předat parametr s hodnotou `false`.

6.2.2 Widget

Pro rozšíření funkcionality gridu o možnost přidávání libovolných komponent je nutné implementovat komponentu, která slouží k jejich vykreslování do šablony gridu. Tato komponenta musí disponovat mechanismy pro mapování vstupů a výstupů komponent. Jako první je důležitý vstupní objekt `options` typu definovaném ve výpisu 6.7, kterým bude popsána komponenta. Na základě konzultace s Markem Konečným [13] byl do objektu pro popis komponenty vložen parametr `id`. Tento parametr slouží k jednoznačné identifikaci dané komponenty a usnadňuje tak její správu a manipulaci při použití v gridu.

```
LgmcGWidgetInput: {
  id: string;
  component: componentType;
  inputs: componentInputs;
}
```

Výpis 6.7: Objekt pro popis komponenty vložené do widgetu slouží k definování typu a vstupů dané komponenty. Objekt se skládá ze tří klíčových parametrů: `id`, `component` a `inputs`. Políčko `id` obsahuje unikátní identifikátor komponenty, `component` určuje typ komponenty a `inputs` slouží k definování vstupů a výstupů komponenty.

Interní implementace

Pro vytvoření nové instance komponenty se použije reference na šablonu `container` a její funkce `createComponent(componentType)` 6.8. Dále je nutné vyfiltrovat pouze vlastnosti,

které jsou přímo definované v objektu `options.inputs` a nejsou zděděné z jiných objektů 6.9. Po vyfiltrování jsou procházeny jednotlivé položky pomocí `forEach()` 6.10 a zjišťuje se, zda se jedná o výstupní či vstupní vlastnosti.

```
const componentRef: ComponentRef =
this.container.createComponent(this.options.component);
```

Výpis 6.8: Vytvoření nové instance komponenty `options.component`. Pro určení pozice vykreslení se používá odkaz do šablony označené pomocí proměnné `container`. A po vytvoření se tato reference uloží do `componentRef`.

```
Object.entries(this.options.inputs).filter(
([propertyName]) =>
  Object.prototype.hasOwnProperty.call(
  this.options.inputs,
  propertyName
  )
)
```

Výpis 6.9: Funkce `entries()` z datového typu `Object` vrací pole vlastností objektu ve tvaru klíč a hodnota. Dále je použita funkce `filter()`, která prochází pole a vrací ho ochuzené o hodnoty, které nesplnily podmínku. `Object.prototype` je základ každého objektu v TypeScriptu a obsahuje důležité funkce. V tomto kódu je využita funkce `hasOwnProperty()`. Tato funkce umožňuje zkontrolovat, zda daná vlastnost `propertyName` patří přímo objektu a není zděděná z jeho prototypového řetězce. Modifikace pomocí `call()` změní objekt ze základního `Object` na `options.inputs`.

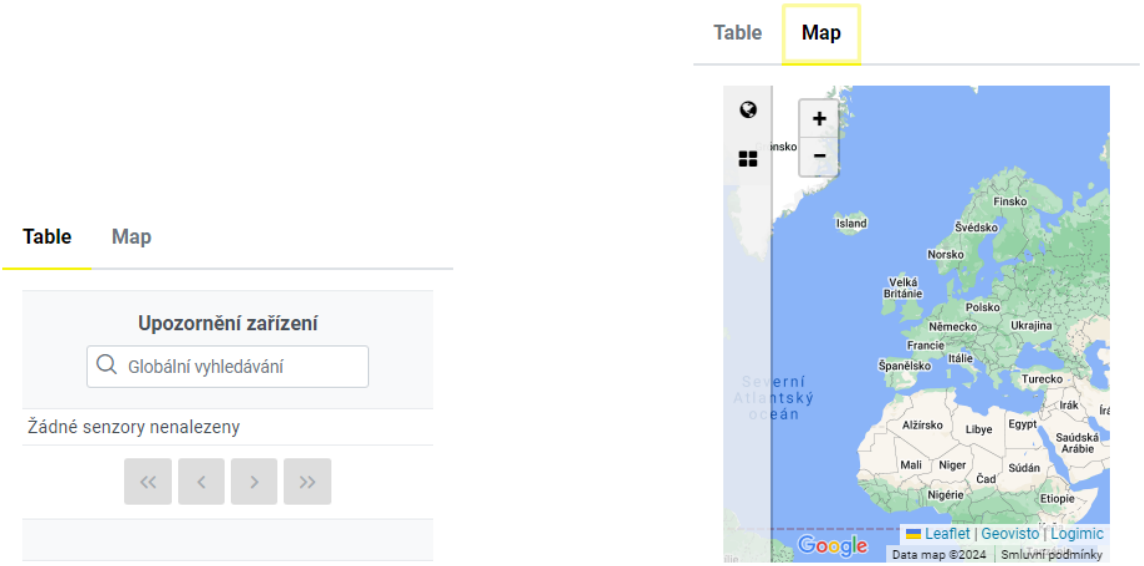
```
    .forEach(([propertyName, value]) => {
       if (
       typeof componentRef.instance[propertyName] === 'object' &&
       componentRef.instance[propertyName] instanceof EventEmitter
       ) {
         componentRef.instance[propertyName].subscribe(data => {
           if (
           typeof this.options.inputs[propertyName] === 'function'
           ) {
             this.options.inputs[propertyName](data);
           }
         });
       } else {
         componentRef.instance[propertyName] = value;
       }
     })
```

Výpis 6.10: Pro každý klíč je třeba zjistit, jestli se jedná o typ `object` a zároveň jestli se jedná o instanci `EventEmitter`, který se používá pro propagaci dat z komponenty do nadřazené komponenty. V případě, že výše uvedená podmínka je splněna, dochází k volání funkce `subscribe()`. Tato funkce umožňuje přihlásit se k odběru událostí emitovaných daným objektem. Jako parametr se uvádí funkce, která se provede vždy, když přijde nová hodnota. V případě že hodnota vlastnosti je typu funkce, dochází k předání dat do funkce definované v objektu `options.inputs`. Jinak se přiřadí hodnota do určité vlastnosti v instanci komponenty.

6.2.3 Responzivní rozložení

Jak je patrné z návrhu prezentovaného na obrázku 5.2, nová komponenta *responzivní rozložení* umožňuje zobrazit dvě libovolné komponenty vedle sebe na větších displejích. V případě menších zobrazovacích zařízení, jako jsou mobilní telefony, se komponenta automaticky přepne do režimu s tlačítky, která slouží k přepínání mezi oběma komponentami. Toto responzivní chování zajišťuje optimální zobrazení obsahu pro všechny typy zařízení. Pro tuto funkcionalitu byla z knihovny PrimeNG využita komponenta `p-tabMenu`, která zobrazuje tlačítka při mobilním zobrazení, jak je vidět na obrázku 6.2.

Dalším klíčovým aspektem komponenty je optimalizace vykreslování obsažených komponent při přepínání tlačítky. To znamená, že se po načtení komponenty již nemění její struktura a obsah při přepínání mezi sloupci. Díky tomu se minimalizuje zátěž prohlížeče a zkracuje doba odezvy při interakci uživatele. Také se některé komponenty (například mapa) nemusejí znovu vykreslovat. Toto je řešeno pomocí stylového předpisu pro třídu `.disabled`, jak je popsáno v kódu 6.11.



(a) Zobrazení tabulky

(b) Zobrazení mapy

Obrázek 6.2: Mobilní zobrazení nové responzivní komponenty s použitím tabulky a mapy.

```
<!--responsive-layout template-->
<p-tabMenu></p-tabMenu>
<!-- left side -->
<div [class.disabled]="selectedIndex !== 0" >
    <ng-content
      select="[leftColContent]"
    ></ng-content>
</div>
<!-- right side -->
<div [class.disabled]="selectedIndex !== 1">
    <ng-content
      select="[rightColContent]"
    ></ng-content>
</div>
```

Výpis 6.11: Zjednodušená šablona *responzivního rozložení* s využitím komponenty `p-tabMenu` a dvou elementů `div` pro zobrazení obsahu na levé a pravé straně. Pro responzivní chování na mobilních zařízeních se aktivní komponenta vybírá na základě hodnoty `selectedIndex`. Vybrané komponentě se pak aplikuje třída `.disabled`, která skryje její obsah (`display: none`). Na webové verzi toto skrytí nenastává, jelikož třída .disabled neobsahuje žádný stylový předpis.

Pro vykreslení dvou komponent se používá Angular element `ng-content`. Tento element slouží jako projekční bod pro obsah komponenty, čímž umožňuje vložit jakoukoliv komponentu do definovaného prostoru v rodičovské komponentě. V případě že je potřeba v komponentě s elementem `ng-content` definovat více projekčních bodů pro vložení obsahu, je nutné je označit pomocí selektorů, jak je vidět v kódu 6.11. A následně je použít

jako direktivu při použití komponenty 6.12. To umožňuje rozlišit mezi nimi a vložit do nich specifický obsah.

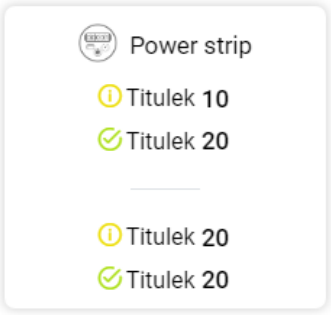
```
<responsive-layout>
    <table #leftColContent></table>
    <map #righColContent></map>
</responsive-layout>
```

Výpis 6.12: Použití komponenty `responsive-layout`, která užívá element `ng-content` a použití direktivy pro identifikaci správného místa pro vložení.

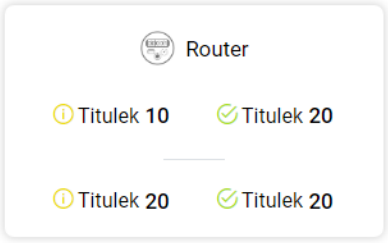
6.2.4 Ostatní nové komponenty

V rámci této práce byla navržena a implementována nová komponenta *karta pro KPI* pro zobrazování *KPI* na kartě. Tato komponenta byla navržena tak, aby mohla být opakovaně použitelná v různých částech aplikace. Kromě vytvoření opakovaně použitelné komponenty pro zobrazování KPI na kartě bylo nezbytné implementovat responzivní design, aby se karta správně zobrazovala v konfigurovatelném gridu 6.2.1. To je důležité hlavně při menších obrazovkách, kde se velikost jednoho pole výrazně zmenší. Responzivní design je vidět na obrázku 6.3. V závislosti na šířce obrazovky se *indikátory KPI* skládají pod sebe, čímž se šetří prostor a zabraňuje se přeplnění. Komponenta *karta pro KPI* požaduje vstupní objekt, který je možné vidět v kódu 6.13. Tento objekt byl definován již v návrhu, ale při implementaci byl rozšířen o:

- `kpi` – model, který firma Logimic interně používá pro ukládání a reprezentaci *KPI*.
- `kpiId` – slouží k jednoznačné identifikaci skupiny KPI indikátorů v rámci aplikace.



(a) Zobrazení karty na malém horizontálním prostoru.



(b) Zobrazení karty na větším prostoru, když už se nemusejí jednotlivé indikátory zobrazovat pod sebou.

Obrázek 6.3: Porovnání mobilního a webového zobrazení pro komponentu *karta pro KPI*.

```
  kpi: any;
  kpiId?: number;
  title: string;
  imageSrc: string;
  kpiIndicators: {
    value?: string|number;
    icon: string;
    icon_color: string;
    title: string;
  }[];
```

Výpis 6.13: Vstupní objekt pro komponentu *karta pro KPI*.

Další nově vytvořenou komponentou je patička stránky (dále jen *footer*). Tato komponenta byla implementovaná v souladu s návrhem. Její vstupy jsou: `footerText` pro spodní volitelný text, `rightUpperText` pro název aplikace, `rightLowerText` a pole akcí `actions` s objektem 6.14.

```
{
    name: string,
    click?: () => void
}
```

Výpis 6.14: Komponenta *footer* může obsahovat tlačítka, které umožňují uživatelům spouštět akce. Pro definování těchto akcí se používá tento objekt, který obsahuje informace o akci, jako je popisek `name` a funkce `click`, která se má provést při stisknutí tlačítka.

6.2.5 Úprava stylů

Pro použití některých komponent v novém konfigurovatelném gridu bylo nutné upravit jejich responzivnost pro různá zobrazení. Stylový předpis těchto komponent byl velice napevno definovaný. Stanovoval pevnou výšku a šířku. Takto pevně definované styly vedly k problémům, jelikož se komponenty nedokázaly automaticky přizpůsobit variabilním rozměrům gridu. Tímto přístupem vznikalo nevzhledné přetékání komponent. Například pro základní komponentu *kruhového indikátoru* (`circle-indicator`) bylo nutné nahradit pevné rozměry, které jsou vidět v části stylového předpisu 6.15 za dynamické předpisy 6.16.

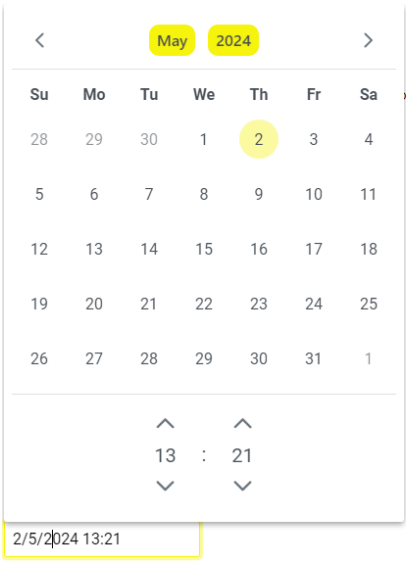
```
.circle-card {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 221px;
  height: 175px;
}
```

Výpis 6.15: Starý styl *kruhového indikátoru*, který byl staticky definovaný.

```
.circle-card {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  height: 100%;
  aspect-ratio: 1/1;
  max-width: 100%;
}
```

Výpis 6.16: Upravené styly pro dynamické zvětšování *kruhového indikátoru*.

Kromě optimalizace responzivity komponent pro konfigurovatelný grid bylo nezbytné zakomponovat do již existujících komponent barevnou paletu aplikace pro jednotný vzhled. Tyto komponenty byly použity z knihovny PrimeNG a doplněny o některé funkční celky. Původní implementace komponent nevěnovala dostatečnou pozornost sladění barev s celkovým designem aplikace, což vedlo k nesourodému a neprofesionálnímu vzhledu. Jako příklad byla v analýze zmíněna komponenta pro *kalendář* a její nová varianta zapadající do aplikace je na obrázku 6.4.



Obrázek 6.4: Komponenta *kalendáře*, která používá barevnou paletu aplikace ACADA.

Globální styly

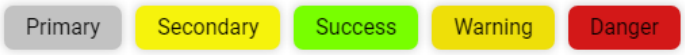
V případě komponent z knihoven (PrimeNG), u kterých se požadují pouze vizuální úpravy a jejich funkčnost se nemění, je efektivnější implementovat globální stylový předpis místo obalování komponent do vlastních komponent. Mezi takovéto komponenty se řadí v aplikaci *tlačítko* a *vyskakovací dialog*.

Tlačítko se nachází v aplikaci ve více barevných provedeních. *Primární* barva se používá pro méně výrazné tlačítka, v tomto případě se jedná o barvu šedou. *Sekundární* barva se používá pro zvýraznění důležitějších akcí pro uživatele. V případě této aplikace se jedná o žlutou barvu. Dále jsou zde barvy *označovací*. Mezi ně patří například zelená nebo červená. Používají se k označení speciálních stavů. Například zelená pro dokončení akce, zatímco červená může upozorňovat na vymazání akce. Poslední důležité barvy jsou *výstražné*. Patří mezi ně například žlutá barva. Používají se pro označení rizikového tlačítka. Mohou mít nevratné následky.

Dále se tlačítka dělí do dvou variant. První z nich je *základní* varianta (na obrázku 6.5), která má bílé pozadí a podle vybraného barevného provedení barevný text. Při najetí na tlačítko se text stane černým a pozadí vezme barvu textu. Druhou variantou jsou tlačítka *barevná* (na obrázku 6.6). Tyto tlačítka mají pozadí dle vybrané barvy. Text je použit černý pro lepší čitelnost.

Každé barevné provedení a každá varianta tlačítka má svoji vlastní *CSS* třídu. Primární barva je jako základní pro všechny tlačítka bez barevné třídy. Další barevné varianty mají třídy ve tvaru `lgmc-button-xxx`. Kde za `xxx` je možné dosadit: *secondary, success, warning*

Obrázek 6.5: Základní tlačítka s bílým pozadím a jejich barevné provedení.



Obrázek 6.6: Barevná tlačítka a jejich barevné provedení.

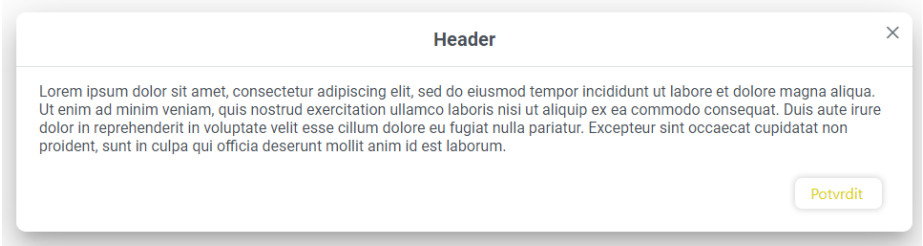
nebo *danger*. Základní varianta je automatická pro každé tlačítko. Pokud je ovšem potřeba barevná varianta, musí její použití obsahovat třídu `lgmc-button-color-fill`. Použití základních a barevných tlačítek je možné vidět v kódu 6.17. Poslední důležité *CSS* třídy jsou pro jejich velikost. Je možné mít menší tlačítko pomocí třídy `lgmc-button-small` a větší tlačítko pomocí třídy `lgmc-button-large`.

```
<!--Basic-->
<p-button label="Primary"></p-button>
<p-button label="Secondary" class="lgmc-button-secondary"
></p-button>

<!--Color-->
<p-button label="Primary" class="lgmc-button-color-fill"
></p-button>
<p-button label="Secondary"
  class="lgmc-button-secondary lgmc-button-color-fill"
></p-button>
```

Výpis 6.17: Jedná se o použití tlačítek `p-button` z knihovny PrimeNG. Na tyto tlačítka jsou aplikované třídy definované v globálním stylovém předpisu.

Komponenta *dialog* v knihovně PrimeNG obsahuje množství vstupů, které by musela vytvořená komponenta zbytečně mapovat. To znamená, že by se mohla komponenta použít samostatně rovnou v aplikaci. Avšak vzhled této komponenty by narušoval konzistentnost celé aplikace. Vzhledem k tomu, že existující komponenta `p-dialog` umožňuje rozsáhlou konfiguraci pomocí stylového předpisu, tak je možné tyto předpisy upravit v globálním stylu. Stačí pro komponentu `p-dialog` použít třídu `lgmc-dialog`. Nový dialog zapadající do stylu aplikace je na obrázku 6.7.



Obrázek 6.7: Nový vzhled dialog obsahující zaoblený design. Šedá čára odděluje hlavičku dialogu a jeho tělo. V patičce dialogu se nachází komponenta pro nové tlačítko.

Kapitola 7

Testování

V rámci vývoje je důležitou součástí i testování výsledků a případná oprava zjištěných chyb. Pro testování výsledků této bakalářské práce bylo využito manuální testování a automatické testy. Na grafických komponentách se ověřuje vizuální vzhled, jako je barevná paleta a rozvržení. Dále je také důležité vyzkoušet její responzivitu na různých typech zařízení. Testování vizuálního vzhledu se provádí manuálně. Responzivita se také může testovat manuálně, ale lze i využít automatické testy. U nově vytvořených komponent nebo upravené logiky komponenty je nutné provést funkční testování, které ověřuje logické chování a funkčnost komponenty dle zadaných specifikací. Funkční testování se provádí pomocí předepsaných automatických testů.

7.1 Manuální testování

Toto testování probíhalo z velké části v rámci implementace. Jedná se hlavně o změny vizuální. Také se testovala funkčnost některých komponent. Z důvodu využití *konfigurovatelného gridu* v bakalářské práci Marka Konečného [13] byla komponenta pro `grid` a `widget` otestována v rámci jeho implementace. Po konzultaci s ním byl upraven vstupní objekt do komponenty `widget` 6.7. Tento objekt byl obohacen o identifikátor, který se používá pro identifikaci jednotlivých instancí. Jako další bylo upraveno rozhraní pro `grid` obohacením o funkce jako například `startEditionGrid` nebo `getComponentsList`. Tyto funkce pomáhají zlepšit komunikaci s komponentou `grid`.

7.2 Automatické testování

Pro automatické testy byla využita bakalářská práce Jána Špačka [27]. Tyto testy jsou postavené na frontend testovacím a automatizačním nástroji Cypress[1]. Je možné tak vytvářet testy pro různé celky aplikace nebo separátně pro každou komponentu zvlášť. Bakalářská práce od Jána Špačka byla zaměřena pouze na jednotlivé stránky aplikace. Byla také testována na staré verzi grafické knihovny, tudíž je možné otestovat správnou funkcionalitu nového *konfigurovatelného gridu* oproti starému zobrazení na řídícím panelu aplikace. V důsledku tohoto testování se zjistilo, že při kliknutí na komponentu *kruhového indikátoru* se neprovede žádná akce, protože v původní verzi řídícího panelu se akce vyskytovala přímo na místě užití komponenty a nebyla zabalena v komponentě.

[1] https://www.cypress.io

7.3 Prezentace firmě Logimic

Během testování jsem úspěšně prezentoval funkční knihovnu pro vedení firmy. Připravené jsem měl praktické ukázky navržených komponent, které demonstrovaly, jak knihovna může rozšířit aplikaci o lepší konfigurovatelnost. Do těchto komponent spadají hlavně: *konfigurovatelná mřížka* a *widget*. Obdržel jsem za ně kladnou zpětnou vazbu a vedení ocenilo potenciál knihovny po zavedení nové funkcionality. Tímto testováním jsem prokázal, že navržená knihovna splňuje požadavky na funkčnost a užitečnost v praxi a zároveň má potenciál stát se cenným nástrojem uživatelů pro zpříjemnění použití aplikace.

Kapitola 8

Závěr

Cílem bakalářské práce bylo vytvořit jednotnou grafickou knihovnu. Tato knihovna má obsahovat základní komponenty, které jsou potřeba pro webové aplikace. Musí také obsahovat komponenty pro vizuální prezentaci dat. Z požadavků firmy Logimic vyplynula potřeba uživatelů přizpůsobovat si některé části aplikace pro jejich lepší užívání. To znamenalo vytvořit komponenty, které tuto konfigurovatelnost budou podporovat.

Jako první jsem se důkladně seznámil s termínem internetu věcí (IoT), jeho důležité součásti a dále s tím související termín chytré město. Zde jsem se zaměřil na využití těchto technologií z pohledu uživatele. Seznámil jsem se také s existujícími aplikacemi v oblasti chytrých měst ve světě a v tuzemsku. Cílem zdokumentovat konkurenční systémy. Na základě práce se sesbíranými daty bylo zapotřebí nastudovat termín vizualizace dat a pro správnou tvorbu komponent i principy uživatelských rozhraní.

Následovala analýza již existující knihovny grafických komponent v aplikaci ACADA v poskytnutém repozitáři od firmy Logimic. Bylo zapotřebí tuto knihovnu projít a označit komponenty, které nesplňovaly požadavky stanovené v zadání. Podle těchto nalezených komponent poté navrhnout upravené komponenty, které by tyto požadavky splňovaly. Také bylo zapotřebí navrhnout nové chybějící komponenty pro konfigurovatelné rozhraní. Na základě návrhů komponent jsem je implementoval v grafické knihovně aplikace ACADA. Pro implementaci byl použit framework Angular založený na jazyce TypeScript.

Již implementované komponenty byly otestovány v rámci vývoje manuálně, ale také pomocí automatických testů z aplikace Cypress. Následně po testování proběhla integrace do platformy ACADA. Když byla integrace funkční, byla tato knihovna prezentována vedení firmy Logimic. Zpětná vazba byla pozitivní a to hlavně z hlediska sjednocení knihovny a vytvoření konfigurovatelnosti pro uživatele. Vývoj této grafické knihovny není zcela dokončen. Bude se dále rozšiřovat o nové konfigurovatelné komponenty.

Literatura

[1] *What is Angular?* 2010. Dostupné z: https://angular.io/guide/what-is-angular.

[2] *LoRaWAN, Sigfox nebo NB-IoT? Srovnání 3 významných typů IoT sítí.* 2020. Dostupné z: https://www.iotport.cz/iot-novinky/lorawan/lorawan-sigfox-nebo-nb-iot-srovnani-3-vyznamnych-typu-iot-siti.

[3] *Dokumentace k řešení Azure IoT Central.* C2024. Dostupné z: https://learn.microsoft.com/cs-cz/azure/iot-central/.

[4] Al Kashoash, H. A. A. a Kemp, A. H. Comparison of 6LoWPAN and LPWAN for the Internet of Things. *Australian Journal of Electrical and Electronics Engineering.* Informa UK Limited. říjen 2016, sv. 13, č. 4, s. 268–274. DOI: 10.1080/1448837x.2017.1409920. ISSN 2205-362X. Dostupné z: http://dx.doi.org/10.1080/1448837x.2017.1409920.

[5] Brooks, K. *How to craft high-quality, reusable and scalable design system components.* 2012. Dostupné z: https://medium.com/alcumus-design/how-to-craft-high-quality-reusable-and-scalable-design-system-components-508ec72c0670.

[6] Carrasco Farré, C., Alcalde, I. a Grimaldi, D. Data analysis, modeling, and visualization in smart cities. In: Elsevier. *Implementing Data-Driven Strategies in Smart Cities: A Roadmap for Urban Transformation.* 2021, s. 173–195.

[7] Chaudhari, B. S., Zennaro, M. a Borkar, S. LPWAN Technologies: Emerging Application Characteristics, Requirements, and Design Considerations. *Future Internet.* MDPI AG. březen 2020, sv. 12, č. 3, s. 46. DOI: 10.3390/fi12030046. ISSN 1999-5903. Dostupné z: http://dx.doi.org/10.3390/fi12030046.

[8] Flanagan, D. *JavaScript: The definitive guide: Activate your web pages.* "O'Reilly Media, Inc.", 2011.

[9] Frenkel, S. *The Rise of Reusable Components.* C 2024. Dostupné z: https://www.bigdropinc.com/blog/the-rise-of-reusable-components/.

[10] Galitz, W. O. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques.* 3rd ed. Canada: Wiley Publishing, 2007. ISBN 978-0-470-05342-3.

[11] Han, Q., Nesi, P., Pantaleo, G. a Paoli, I. Smart city dashboards: design, development, and evaluation. In: IEEE. *2020 IEEE International Conference on Human-Machine Systems (ICHMS).* 2020, s. 1–4.

[12] Khan, R., Khan, S. U., Zaheer, R. a Khan, S. Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges. In: IEEE. *2012 10th International Conference on Frontiers of Information Technology*. Prosinec 2012, s. 257–260. DOI: 10.1109/fit.2012.53. ISBN 978-0-7695-4927-9. Dostupné z: http://dx.doi.org/10.1109/fit.2012.53.

[13] Konečný, M. *Systém pro uživatelskou konfiguraci platformy Smart City*. 2024. Bakalářská práce. Vysoké učení technické v Brně, Brno. Vedoucí práce Ing. Jiří Hynek, P.

[14] Mehmood, Y., Ahmad, F., Yaqoob, I., Adnane, A., Imran, M. et al. Internet-of-things-based smart cities: Recent advances and challenges. *IEEE Communications Magazine*. IEEE. 2017, sv. 55, č. 9, s. 16–24.

[15] Mekki, K., Bajic, E., Chaxel, F. a Meyer, F. A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT Express*. Elsevier BV. březen 2019, sv. 5, č. 1, s. 1–7. DOI: 10.1016/j.icte.2017.12.005. ISSN 2405-9595. Dostupné z: http://dx.doi.org/10.1016/j.icte.2017.12.005.

[16] Mohanty, S. P., Choppali, U. a Kougianos, E. Everything you wanted to know about smart cities: The Internet of things is the backbone. *IEEE Consumer Electronics Magazine*. IEEE. 2016, sv. 5, č. 3, s. 60–70.

[17] Mukhopadhyay, S. C. a Suryadevara, N. K. *Internet of things: Challenges and opportunities*. Springer, 2014.

[18] Nacheva, R. et al. Principles of user interface design: important rules that every designer should follow. *IZVESTIA, JOURNAL OF THE UNION OF SCIENTISTS-VARNA, ECONOMIC SCIENCES SERIES*. 2015, sv. 1, s. 140–149.

[19] Navani, D., Jain, S. a Nehra, M. S. The Internet of Things (IoT): A Study of Architectural Elements. In: IEEE. *2017 13th International Conference on Signal-Image Technology, Internet-Based Systems (SITIS)*. Prosinec 2017. DOI: 10.1109/sitis.2017.83. Dostupné z: http://dx.doi.org/10.1109/sitis.2017.83.

[20] Palaniappan, R., John, T. J. a Nagaraj, V. IoT Solutions for Smart Cities. In: IOP Publishing. *IOP Conference Series: Materials Science and Engineering*. 2020, sv. 955, č. 1, s. 012004.

[21] Picioroaga, I.-I., Eremia, M. a Sanduleac, M. SMART CITY: Definition and Evaluation of Key Performance Indicators. In: IEEE. *2018 International Conference and Exposition on Electrical And Power Engineering (EPE)*. říjen 2018, s. 217–222. DOI: 10.1109/icepe.2018.8559763. ISBN 978-1-5386-5062-2. Dostupné z: http://dx.doi.org/10.1109/icepe.2018.8559763.

[22] Rose, K., Eldridge, S. a Chapin, L. The internet of things: An overview. *The internet society (ISOC)*. Reston, VA. 2015, sv. 80, s. 1–50.

[23] Sharma, V. a Tiwari, A. K. A study on user interface and user experience designs and its tools. *World Journal of Research and Review (WJRR)*. 2021, sv. 12, č. 6, s. 41–45.

[24] Talari, S., Shafie khah, M., Siano, P., Loia, V., Tommasetti, A. et al. A Review of Smart Cities Based on the Internet of Things Concept. *Energies*. MDPI AG. březen 2017, sv. 10, č. 4, s. 421. DOI: 10.3390/en10040421. ISSN 1996-1073. Dostupné z: http://dx.doi.org/10.3390/en10040421.

[25] Weber, R. H. a Weber, R. *Internet of things*. Springer, 2010.

[26] Zanella, A., Bui, N., Castellani, A., Vangelista, L. a Zorzi, M. Internet of things for smart cities. *IEEE Internet of Things journal*. Ieee. 2014, sv. 1, č. 1, s. 22–32.

[27] Špaček, J. *Optimalizace a testování frontendové části na platformě Smart City*. 2024. Bakalářská práce. Vysoké učení technické v Brně, Brno. Vedoucí práce John, I. P.