FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Diplomová práce

Online vizualizace dat z building management systémů

Studijní program: N0613A140028 Informační technologie
Autor práce: **Bc. Václav Kesler**
Vedoucí práce: Ing. Jan Kraus, Ph.D.
Ústav mechatroniky a technické informatiky

Liberec 2023

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání diplomové práce

Online vizualizace dat z building management systémů

Jméno a příjmení:	**Bc. Václav Kesler**
Osobní číslo:	M21000160
Studijní program:	N0613A140028 Informační technologie
Zadávající katedra:	Ústav mechatroniky a technické informatiky
Akademický rok:	2023/2024

Zásady pro vypracování:

1. Seznamte se s možnostmi tvorby moderních grafických webových aplikací s využitím aktuálních frameworků a nástrojů pro vývoj uživatelsky přívětivé interaktivní vizualizace dat z chytré budovy.
2. Vyberte vhodné nástroje pro implementaci jednoduchého ukázkového back-end a komplexního front-end řešení takové vizualizace.
3. Aplikaci implementujte, důkladně otestujte a ověřte její výkonnostní a jiné limity.
4. Na vhodně zvolených příkladech demonstrujte realizované vizualizace a jejich interaktivní funkce.
5. V závěru práce stručně shrňte dosažené výsledky a diskutujte možnosti dalšího rozvoje tématu.

Rozsah grafických prací:	dle potreby dokumentace
Rozsah pracovní zprávy:	40 až 50 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

[1] HIMSCHOOT, Peter. Microsoft Blazor. Online. Berkeley, CA: Apress, 2020. ISBN 978-1-4842-5927-6. Dostupné z: https://doi.org/10.1007/978-1-4842-5928-3. [cit. 2023-10-01].

[2] HAAS, Andreas; ROSSBERG, Andreas; SCHUFF, Derek L.; TITZER, Ben L.; HOLMAN, Michael et al. Bringing the web up to speed with WebAssembly. Online. In: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY, USA: ACM, 2017, s. 185-200. ISBN 9781450349888. Dostupné z: https://doi.org/10.1145/3062341.3062363. [cit. 2023-10-01].

[3] LITVINAVICIUS, Taurius. Exploring Blazor. Online. Berkeley, CA: Apress, 2019. ISBN 978-1-4842-5445-5. Dostupné z: https://doi.org/10.1007/978-1-4842-5446-2. [cit. 2023-10-01].

[4] MARTIN, Robert C. Clean code: a handbook of agile software craftmanship. Robert C. Martin series. Upper Saddle River, NJ: Prentice Hall, c2009. ISBN 0-13-235088-2.

[5] SCHWARZMÜLLER, Maximilian. React Key Concepts. 1. Packt Publishing, 2022. ISBN 978-1-80323-450-2.

Vedoucí práce:	Ing. Jan Kraus, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	12. října 2023
Předpokládaný termín odevzdání:	14. května 2024

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Černohorský, Ph.D.
vedoucí ústavu

V Liberci dne 12. října 2023

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

9. listopadu 2023 Bc. Václav Kesler

ONLINE VIZUALIZACE DAT Z BUILDING MANAGEMENT SYSTÉMŮ

ABSTRAKT

Tato diplomová práce se zabývá online vizualizací dat z building managment systémů. V úvodu práce je představen současný stav problematiky, včetně příkladů z praxe. Následně jsou analyzovány dostupné nástroje pro vývoj a tvorbu uživatelského rozhraní. V části o realizaci je nejprve nastíněna struktura řešení, která se skládá ze dvou částí: vizualizační aplikace a ukázkové aplikace reprezentující server budovy. Jako primární vývojový nástroj byl zvolen framework Next.js, který je postavený na popuární knihovně React. Hlavním stavebním kamenem vizuální části řešení je design systém Material Design, který je implementován pro React knihovnou Material UI. Obě aplikace jsou nasazené na platformě Vercel a jako databázi využívají MongoDB, která běží na cloudovém řešení MongoDB Atlas. Vizualizační aplikace umožňuje uživateli po přihlášení přidávat koncové body, které slouží jako zdroje dat. Uživatel dále může vytvářet dashboardy, které fungují jako místa, kam lze umístit grafy a tabulky a seskupit je do logických celků. V rámci dashboardu je možné vytvořené vizualizace přesouvat a upravovat jejich velikost podle potřeby. Je také možné měnit zobrazovaný časový interval. Uživatel má k dispozici sloupcový graf, spojnicový graf, tabulku a interaktivní kombinaci tabulky s grafem. V části týkající se ukázkové aplikace pro budovy je definováno rozhraní API, které umožňuje komunikaci s hlavní aplikací. Tato ukázková aplikace zahrnuje funkce pro generování testovacích dat a sběr reálných dat. Sbírána jsou data o proudu a napětí z chytré zásuvky a také aktuální teplota v Praze. Obě aplikace byly otestovány. Byla ověřena funkčnost hlavní aplikace v různých prohlížečích a na obrazovkách s různou velikostí. Výkonnost aplikace společně s dalšími parametry byla ověřena nástrojem Lighthouse. Aplikace splňuje standardy firmy Google pro výkonnou a přístupnou webovou aplikaci.

Klíčová slova: Building management systémy, Online vizualizace dat, React, Next.js, Material Design

ONLINE BMS DATA VISUALIZATION

ABSTRACT

This master's thesis deals with the online visualization of data from building management systems. The introduction presents the current state, including real world examples. Following that, the analysis explores the available tools for development and the creation of user interfaces. In the implementation section, the solution structure is outlined, consisting of two parts: a visualization application and example application representing the building server. The Next.js framework, which is built on the popular React library, was chosen as the primary development tool. The main building block of the visual part of the solution is Material Design, implemented for React with the Material UI library. Both applications are deployed on the Vercel platform and use MongoDB as the database, running on the MongoDB Atlas cloud solution. The visualization application allows users to add endpoints after logging in, which serve as data sources. Users can also create dashboards, which function as places to place graphs and tables and group them into logical units. Within the dashboard, created visualizations can be moved and resized as needed. It is also possible to change the displayed time interval. The user has access to a bar chart, a line chart, a table, and an interactive combination of a table with a graph. In the section about the example building application, an API interface is defined, which allows communication with the main application. This example application includes functions for generating test data and collecting real data. Information on the current flow and voltage from a smart socket is being collected, alongside the present temperature in Prague. Both applications have been tested. The functionality of the main application has been verified in various browsers and on screens of different sizes. The application's performance, along with other parameters, was verified by the Lighthouse tool. The application meets Google's standards for a performant and accessible web application.

Keywords: Building managment systes, Online data visualization, React, Next.js,Material Design

PODĚKOVÁNÍ

Rád bych poděkoval panu Ing. Janu Krausovi, Ph.D. za vedení práce. Poděkovat bych také chtěl i své rodině za neochvějnou podporu po dobu celého studia.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK

API	Application programming interface
CDN	Content delivery network
CI/CD	Continuous integration / Continuous delivery
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	Hypertext Markup Language
JS	JavaScript
NPM	Node.js package manager
PR	Pull request
SEO	Search engine optimization - optimalizace pro vyhledávače
UI	User interface - uživatelské rozhraní
UX	User experience - uživatelská zkušenost
W3C	World Wide Web Consortium
WASM	WebAssembly

1 Úvod

Téma chytrých budov je aktuální zejména v kontextu dnešní doby, kdy se neustále zvyšují náklady na energie a provoz. Online aplikace pro vizualizaci dat pak slouží jako nástroj k přístupu k aktuálním údajům o budově odkudkoliv na světě a také jako užitečný pomocník pro mapování míst, kde lze snížit náklady na provoz.

Tato práce si klade za cíl vytvořit online aplikaci pro vizualizaci dat, která komunikuje s ukázkovým serverem umístěným v budově. Vytvořená aplikace by měla být jednoduchá na používání jak pro uživatele, tak pro správce budovy.

Úvodní část práce obsahuje seznámení s aktuálním stavem problematiky. Zde je popsáno, co jsou building management systémy a k čemu slouží. Tato část také obsahuje příklady aktuálně dostupných řešení. Dále jsou zkoumány možnosti vývoje webových rozhraní. Jsou zde zmíněny aktuální trendy mezi vývojovými nástroji a možnosti tvorby vizuálního uživatelského rozhraní. Testování softwaru je nedílnou součástí vývoje, a proto je zařazena i část věnovaná problematice testování.

Následující část se zabývá samotnou realizací projektu. Nejprve je popsána struktura řešení, následuje část věnovaná výběru nástrojů pro vývoj aplikace. Řešení se skládá ze dvou částí: částí serveru na straně budovy a částí aplikace na straně uživatele.

Hlavní část práce se zabývá vytvářením aplikace pro vizualizaci dat na straně uživatele či správce budovy. V tomto oddílu jsou popsány dostupné interaktivní prvky pro vizualizaci dat a všechny možnosti aplikace. Je zde také zdůrazněno zabezpečení proti neoprávněnému přístupu, což je klíčové při práci s reálnými daty.

V části o ukázkové aplikaci pro budovy je nejprve definován způsob komunikace s hlavní aplikací. Poté následuje část, která se zabývá procesem generování testovacích dat a zpracováním reálných dat.

V závěrečné části práce jsou prezentovány dosažené výsledky a diskutovány případné problémy, které se vyskytly během vývoje. Jsou zde rovněž nastíněny možnosti dalšího rozšíření a vývoje navrženého řešení.

2 REŠERŠE

2.1 STÁVÁJÍCÍ STAV PROBLEMATIKY

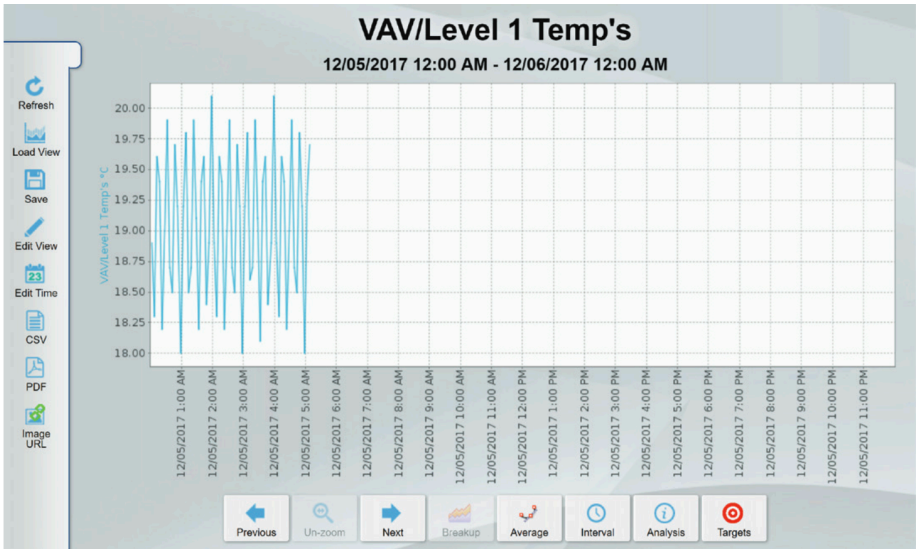
Building management system (BMS) neboli česky systém pro správu budov označuje komplexní systém, který usnadňuje správu budov v širokém spektru oblastí. Tento systém slouží k efektivnímu řízení a monitorování různých aspektů budovy, včetně správy energií, inteligentních zařízení, kontrole přístupu do budovy nebo například plánování údržby. Úkolem takovýchto systémů je umožnit řízení budovy z jednoho místa, snížit náklady na provoz budovy, snížit administrativní náročnost a zvýšit kvalitu života jejím obyvatelům ale i správcům. Důležité je také zmínit, že ne všechny systémy pro správu budov jsou rovnocenné. Některé systémy se zaměřují spíše na obytné budovy s důrazem na správu obyvatel a jiné se zaměřují na správu průmyslových objektů.

Na trhu existují komplexní řešení, která se starají o kompletní správu budov. Firmy nabízející tato řešení většinou nabízí společně se softwarem pro správu budov samotnou integraci jednotlivých prvků. Vždy záleží na konkrétních požadavcích majitele budov a současném stavu budovy.

2.1.1 Příklady dostupných softwarů na trhu

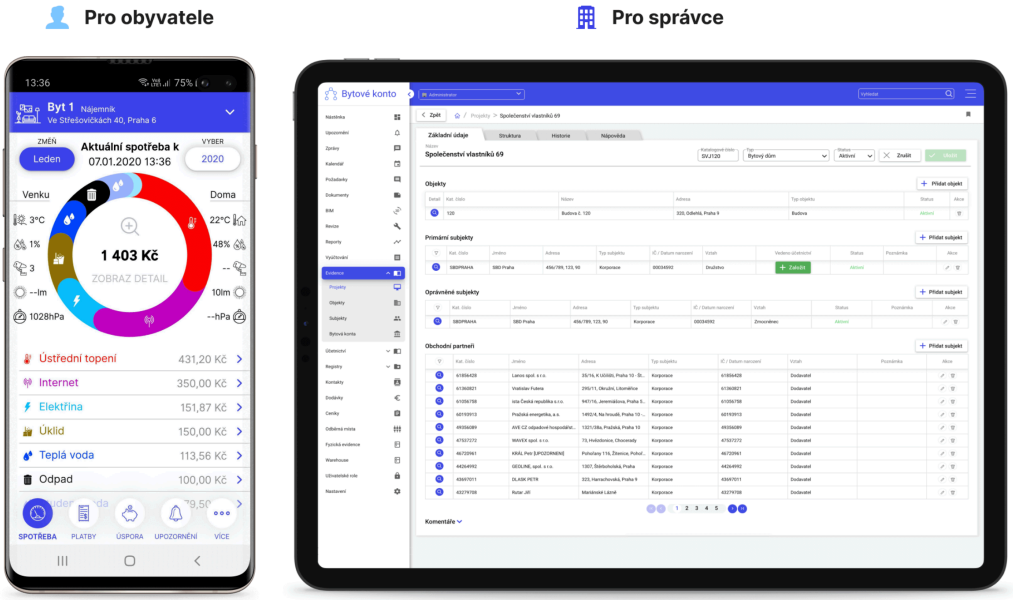
Příkladem nástrojů pro správu spíše průmyslových objektů může být software od australské firmy Optergy. Firma nabízí software, za pomoci kterého, je možné plně spravovat budovu. Lze komunikovat se zařízeními pomocí standardů BACnet nebo MODBUS. Je možné vytvářet aplikační logiku, nastavovat alarmy, ale například i zobrazovat data (obr. 2.1). Prostředí je navrženo jako nástroj pro integrátory, kteří zde vytvoří prostředí pro koncové uživatele pomocí nástroje „DisplayTool“. [6]

Na rozdíl od firmy Optergy se česká firma BeIT zaměřuje spíše na nástroje pro řízení bytových domů. (obr. 2.2). Jejich software bytové konto [7] je rozdělený na část pro obyvatele a správce. Obyvatelé mají k dispozici grafiky s aktuálními spotřebami, mohou komunikovat se správcem či spravovat své platby. Správce má možnost například plánovat revize, spravovat dokumenty nebo třeba nastavovat uživatelské role. Software může obsahovat i 3D vizualizaci objektu s jednotlivými chytrými prvky jako jsou například chytré plynoměry či elektroměry.



Obrázek 2.1: Zobrazení dat pomocí softwaru od firmy Optergy [6]

V obou případech aplikace obsahují kromě vizualizace dat také spoustu funkcionalit navíc, které byly vytvořeny na míru konkrétním potřebám. Nicméně faktem zůstává, že obě aplikace by nebyly úplné bez vizualizace, která je tedy nedílnou součástí každého BMS.



Obrázek 2.2: Bytové konto firmy BeIT [35]

2.2 Vývojové nástroje

Podle statistik z roku 2023 zveřejněných na GitHubu [9], je JavaScript stále nejpoužívanějším programovacím jazykem v open source projektech. Jedním z důvodů tohoto trendu může být jeho nativní podpora v prohlížečích. [2] Neznamená to však, že je to jediná možnost při vývoji uživatelských rozhraní pro web. Díky existenci WebAssembly zkráceně WASM či staršího asm.js [11], je pro vývoj na webu možnost použít i jiné programovací jazyky jako například C++, Go nebo Rust.

2.2.1 WebAssembly

WebAssembly je binární instrukční formát určený pro zásobníkové virtuální stroje. [8]. Byl navržen jako vhodnější cíl kompilace (compilation target), než je JavaScript, do jehož subsetu kompiluje výše zmíněný asm.js. Hlavní požadavky při vytváření WASM byly rychlost, přenositelnost, bezpečnost a kompaktnost. [2] Experimenty ukazují, že WASM může být více než 20x rychleji nativně dekódován než JavaScript parsován. [8]

WASM je podporován ve čtyřech hlavních moderních prohlížečích (Chrome, Safari, Firefox a Edge). [8] Na webu „caniuse.com“ je možné zjistit přesné verze prohlížečů, od kterých je WASM podporován [18]. Zde lze rovněž vidět, že WASM je v současnosti podporován nejenom v hlavních prohlížečích, ale i v mobilních prohlížečích, jako je například Samsung Internet. Celková podpora WASM je 97,14 % všech současných zařízení, což vyplývá ze statistik dostupných na webu statcounter [19], který sbírá data z více než 1,5 milionu webových stránek.

Při výběru programovacího jazyka je nutné zhodnotit vhodnost pro danou aplikaci. Přímo na stránkách WebAssembly je výčet vhodných situací pro použití WASM. Jedná se zejména o výpočetně náročné operace, jako je například úprava videa, hry nebo třeba rozpoznávání obrazu. [8]

WebAssembly je navržené tak, aby JavaScript spíše doplnilo a ne ho nahradilo. [8] Není nutné ho tedy použít jako hlavní platformu při vývoji, ale jako doplněk pro výpočetně náročné operace, kde je vhodný. Vizualizace dat se zejména zabývá zobrazením dat pomocí tabulek a grafů, pro které je použití JavaScriptu dostatečné. WASM je v rámci vizualizační aplikace vhodný kandidát na použití například při zpracování velkého množství dat nebo může být využit třeba pro implementaci podobné funkcionality, jako bylo zmíněno v kapitole 2.1.1, a to 3D zobrazení budovy z plánů.

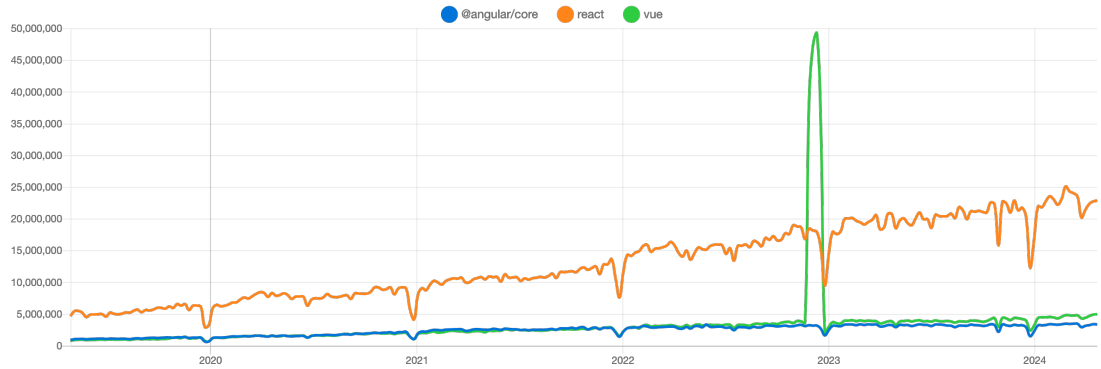
2.2.2 JavaScriptové frameworky

Je mnoho způsobů, jak nahlížet na popularitu jednotlivých frameworků pro JavaScript. Mohou to být například průzkumy prováděné různými společnostmi. Jednou takovou společností je česká firma JetBrains, která kaž-

doročně provádí průzkum mezi uživateli, kteří si jako hlavní programovací jazyk zvolili JavaScript nebo TypeScript.

V posledním průzkumu provedeném v roce 2023 odpovědělo celkově 26 348 respondentů. [22]. Otázkou bylo, jaké všechny frameworky či knihovny respondenti použili za posledních dvanáct měsíců. Mezi odpovědmi byly i jiné než frontendové knihovny například Express nebo frameworky jako Next.js, které jsou postavené na knihovně React. Tyto odpovědi byly záměrně v následujícím výčtu vynechány. Mezi třemi nejpoužívanějšími knihovnami či frameworky se na prvním místě umístil s 57 % React, na druhém místě s 32 % se umístilo Vue a třetí příčku obsadil s 20 % Angular.

Dalším způsoben, jak hodnotit popularitu je hodnocení podle počtu stažení pomocí správce balíčků pro JavaScript NPM (Node.js package manager). Na obrázku 2.3 je vidět vývoj za posledních 5 let. React obsadil s poměrně velkým náskokem první místo. Vue v posledních letech nabývá na popularitě a během posledního třech let se drží na druhé příčce v počtu stažení. Angular stejně jako v průzkumu od společnosti Jetbrains obsadil třetí místo. Kolem prosince 2022 je možné v grafu vidět obrovský nárůst v počtu stažení Vue, který byl ale způsobený chybou v automatizaci, a nikoliv náhlým zájmem z řad vývojářů.



Obrázek 2.3: Počet stažení pomocí NPM za posledních 5 let [23]

Tyto přehledy a statistiky jsou dobrým indikátorem hlavního proudu ve webovém vývoji. Použití jednoho ze tří největších frameworků/knihoven je jistou zárukou velké komunity vývojářů a budoucí podporou v následujících letech. Vždy ale může přijít revoluční technologie, která změní rozložení na trhu a bude pro vývoj uživatelských rozhraní vhodnější. Tyto frameworky/knihovny mají však mnohem menší šanci ze dne na den zaniknout a ukončit budoucí vývoj než nové menší projekty.

2.2.3 Back-end využívající JavaScript

Od roku 2009 JavaScript přestal být výhradně skriptovacím jazykem pro frontendový vývoj. Díky platformě Node.js se stal užitečným nástrojem i pro

vývoj backendových aplikací. Node.js poskytuje meziplatformové běhové prostředí, které umožňuje provádění JavaScriptového kódu mimo kontext prohlížeče, tedy přímo na serveru nebo v počítači. Přestože Node.js postrádá některá rozhraní typická pro prohlížeč, jako jsou manipulace s DOM (Document Object Model), nabízí širokou škálu knihoven například pro práci se souborovými systémy. Node.js byl vytvořen, tak aby byl výkonný a dobře škálovatelný. [39]

Pro tvorbu backendových řešení je možné zvolit čisté Node.js bez žádné další knihovny, avšak tato volba není optimální z hlediska časové náročnosti. Implementace všech funkcionalit vyžaduje rozsáhlou práci programátora. Z tohoto důvodu vznikly různé knihovny a frameworky pro usnadnění vytváření komplexních řešení. Jedním z nejpopulárnějších backendových frameworků pro JavaScript je Express. [22].

Express, ve srovnání s použitím čistého Node.js, nabízí vlastní mechanismus routování, který pomáhá snížit množství opakujícího se kódu. Dále umožňuje definování tzv. middleware, což jsou funkce prováděné mezi příchozím požadavkem a odchozí odpovědí. Tyto funkce mohou zahrnovat například poskytování statických souborů, zpracování chyb, či kompresi HTTP odpovědí. [39]

Express je framework, který je nazaujatý (unopinionated), což znamená, že neklade mnoho omezení na to, jak propojit různé komponenty k dosažení cíle, ani na výběr konkrétních komponent. To umožňuje programátorovi například volně skládat middleware a rozhodovat se o jejich pořadí.

Dalším backendovým frameworkem je Next.js. Ten je oproti Expressu zaujatý (opinionated). To znamená, že má přesně stanovená pravidla, například ohledně vytváření routování, které se v případě Next.js odvíjí od struktury složek a souborů. [24] Použití zaujatého frameworku může mít nevýhodu v menší flexibilitě v řešení, avšak tato nevýhoda je kompenzována rychlostí vývoje, zejména pokud se jedná o standardizovaný problém, který je frameworkem dobře definován.

2.3 VIZUÁLNÍ STRÁNKA

Hlavním nástrojem pro úpravu vizuální části webové stránky jsou kaskádové styly neboli zkráceně CSS. Hlavním smyslem CSS bylo oddělit vzhled dokumentu od jeho struktury a obsahu. Kaskádové styly umožňují úpravu například fontů, barev nebo třeba odsazení. CSS je standard spravovaný pracovní skupinou zaměřenou na CSS, která je součástí W3C (World Wide Web Consortium), organizace vyvíjející webové standardy. [30]

Tato pracovní skupina je složena ze zástupců firem, které vytváří samotné prohlížeče a dalších společností, které mají zájem o CSS. Jsou také přizváni nezávislí experti, kteří nemají přímou vazbu na členy organizace. Přestože v této pracovní skupině jsou zástupci jednotlivých firem vyvíjející prohlížeče, tak je implementace a podpora nových funkcí napříč prohlížeči rozdílná.

Příkladem může být například „container query“, která umožňuje aplikovat styly na základě velikosti kontejneru, ve kterém se prvek nachází. Dříve bylo možné používat pouze velikost celé obrazovky jako zdroj informace o velikosti. Tato funkcionalita byla implementována ve většině prohlížečů v druhé polovině roku 2022, ale ve Firefoxu začala být podporována až ve verzi 110, která byla vydána až v únoru 2023. Tato funkcionalita byla definována v pracovním návrhu „CSS Containment Module Level 3“, jehož první verze byla publikována už 2. listopadu 2021 a poslední pak 18. srpna 2022. [33]

Na tomto příkladu je vidět, že může trvat více než rok mezi prvním oficiální návrhem a samotnou implementací v prohlížečích. To ale není posledním krokem, tím je až samotné použití funkcionality vývojáři na webu. Vývojáři musí upravit své stávající postupy při vývoji a je nutné, aby i komponentní knihovny, které jsou použity vývojáři, tuto funkcionalitu podporovaly. To může zabrat i více času než samotná doba od návrhu standardu po implementaci v prohlížečích. Například komponentní knihovna MUI v současné době „container query“ stále přímo nepodporuje, přestože je to více než rok od zahájení podpory ve všech velkých prohlížečích. [25]

Nové funkcionality však nejsou tím jediným, co se v oblasti kaskádových stylů vyvíjí. Vznikají i nové přístupy k tomu, jak samotné CSS efektivně psát. Jedním z novějších přístupů může být takzvaný „utility first“. Namísto použití tradičních sémantických tříd, ve kterých je definováno, jak má daná komponenta vypadat, jsou použity takzvané utility třídy. Tyto třídy se zaměřují pouze na jednu specifickou věc, tím může být změna barvy nebo třeba velikosti odsazení.

Tento přístup využívá CSS framework Tailwind [26]. Výhodou tohoto přístupu je, že i po delší době je na první pohled jasné, jak má komponenta vypadat. Není nutné nahlížet do dalšího souboru, kde jsou styly definovány. To může být výhodou i pro nové vývojáře pracující na projektu. Stačí jim znalost Tailwindu a nemusí se jako v tradičním projektu seznamovat se sémantickými třídami, které jsou typické pro daný projekt. Tato výhoda může být vnímaná ale i jako nevýhoda, jelikož pro použití Tailwindu je nutné se na začátku naučit jména utility tříd, které reprezentují dané vlastnosti a nestačí pouhá znalost kaskádových stylů.

V rámci vytváření kaskádových stylů v React projektu existují i další možnosti. Kromě již zmíněného Tailwindu, lze také použít jedno z řešení, které používá zápis stylů přímo v JavaScriptovém souboru. Mezi knihovny implementující CSS v JavaScriptu patří například styled-components [29] nebo knihovna Stylex od společnosti Meta [28].

Pro psaní kaskádových stylů lze také využít některý z preprocessorů. Preprocesory mají vlastní syntaxi a rozšiřují základní funkcionalitu kaskádových stylů. Tato syntaxe je pak kompilována nebo interpretována do CSS. Jedním z populárních preprocesorů je SASS (Syntactically Awesome Style Sheets). [27] SASS je plně kompatibilní se všemi verzemi CSS a rozšiřuje základní funkcionalitu například o vnořování, funkce nebo mixiny. Tyto funkcionality pak umožňuji vytvářet přehlednější strukturu, která je jednodušší na spravo-

vání než samotné CSS. Na rozdíl od Tailwindu se není nutné učit celé nové názvy, ale pouze funkcionality, o které je CSS rozšířeno. Použití preprocesoru sebou ale také nese jistá úskalí. Například je nutné mít na paměti, že po kompilaci může vzniknout zbytečně velký CSS soubor kvůli nevhodnému použití vnořování. Přílišné vnořování totiž generuje dlouhé selektory, kterým bylo možné se vyhnout.

2.3.1 Design Systémy

Další možností, jak vytvořit vizuálně přívětivou stránku, je použití design systému. Design systém je set stavebních bloků a standardů, který pomáhá udržet konzistentní uživatelský zážitek napříč webovou stránkou či aplikací. Design systémy jsou většinou navrhovány grafickými designéry, kteří tyto systémy vytváří v programech jako je například Figma. Na základě těchto návrhů pak v jednotlivých programovacích jazycích vznikají samotné implementace navržených prvků. Tyto implementované prvky pak dohromady tvoří komponentní knihovny.

Každá webová stránka či aplikace by měla mít alespoň minimální design systém, ve kterém jsou definovány fonty, barvy a tlačítka. Ty lepší design systémy pak obsahují i další komponenty a pravidla, které jsou napříč aplikací či webem přepoužívány. Využitím navrženého design systému se kromě udržení konzistentnosti zvyšuje i rychlost vývoje a usnadňuje následnou údržbu.

Existují firmy, které nabízí své design systémy dalším designérům a vývojářům k použití. Dokonce i Česká republika má volně dostupný design systém, který má i vlastní implementaci komponent. Tento design systém musí používat například stránky ministerstev nebo ústřední orgány státní správy. [31]. Mezi další design systémy, které jsou dostupné, patří například Material Design, Ant Design nebo třeba DevExpress.

Material Design [13] je open source design systém vyvíjený firmou Google. V současné době je ve verzi 3. Tento design systém zahrnuje podrobné instrukce z oblasti UX (user experience) a implementace UI (user interface) komponent určené pro Android, Flutter a Web. Součástí je také design kit pro Figmu. Existují i neoficiální implementace tohoto design systému speciálně přizpůsobené konkrétním programovacím jazykům nebo frameworkům. Například pro jazyk Blazor existuje komponentní knihovna s názvem Matblazor. [32] V případě Reactu je nejpopulárnější implementací MaterialUI, zkráceně MUI [25]. Rozdíl mezi těmito dvěma konkrétními implementacemi spočívá nejen v programovacím jazyce, ale také v různých aspektech, jako je množství implementovaných komponent nebo animace. MUI například zahrnuje komponentu nazvanou „stepper“, která byla součástí Material Designu verze jedna, ale v současné verzi tři již není dokumentována. Přesto MUI tuto komponentu nadále podporuje. Rozdíl v animacích je dobře patrný například při použití komponenty „drawer“. Animace v implementaci MUI působí plynuleji ve srovnání s implementací MatBlazor.

Jako další z populárních design systému je Ant Design. [15] Je vyvíjen firmou Ant Group, která vlastní například portál Alibaba. Ant Design je po Material Designu druhým největším design frameworkem pro React. Kromě implementace pro React existují implementace také pro Vue, Angluar a Blazor. Současná verze Ant Designu je pět. Samozřejmostí je také podpora Figmy a rozsáhlá dokumentace. Oproti Material Designu však dokumentace nemá podrobně popsanou přístupnost jednotlivých komponent. Přístupnost znamená, že komponenty byly navrhovány a vyvíjeny tak, aby je mohli používat i lidé s handicapy.

Další design systém známý především v komunitě C# vývojářů je DevExpress [14]. DevExpress je vhodnou volbou v případě vývoje desktopové aplikace v C# a webové aplikace v Blazoru nebo jiném frontendovém frameworku, jelikož umožňuje zachovat stejnou vizuální stránku pro obě aplikace. Kromě Blazoru jsou nabízeny i implementace pro React, Vue, Angular a dokonce i JQuery. Součástí dokumentace je i UI kit pro Figmu, který ale není v současnosti 100% hotový. Na rozdíl od dvou předešlých design systémů je DevExpress placenou knihovnou.

Jak už bylo zmíněno na začátku kapitoly, použití design systému urychluje vývoj a usnadňuje pozdější údržbu projektu. V pozdější fázi vývoje, kdy je součástí týmu i grafický designér, je jednodušší upravit vizuální stránku aplikace zejména díky existenci podkladů ve Figmě a možnosti vytváření takzvaných témat. Témata jsou způsob, jak upravovat globálně fonty, barvy a další parametry. Designér už má od začátku k dispozici set komponent, které jsou použity i na webové stránce a může je rovnou začít upravovat. Pozdější zásah do vizuální stránky není tedy tak náročný jako v případě vývoje vlastních komponent.

2.4 TESTOVÁNÍ

Testování softwaru je proces, při kterém se kontroluje kvalita, funkčnost a výkon ještě před nasazením do produkčního prostředí. Existují dvě možnosti, jak psát testy. První přístup spočívá v psaní testů po napsání kódu a druhý přístup spočívá v psaní testů před napsáním kódu.

Test driven development zkráceně TDD neboli programování řízené testy je přístup k vývoji softwaru, kdy jsou nejdříve napsány jednotkové testy. Tyto testy jsou nejprve neúspěšné a až na jejich základě se tvoří samotná implementace funkcionality, pro kterou dané testy projdou jako splněné.

Mít testovací prostředí umožňuje programátorům provádět změny v kódu beze strachu, že něco rozbijí. Pokud se totiž něco rozbije, testy by měly selhat. Z toho plyne, že testy by měly pokrývat veškeré krajní scénáře, ke kterým může dojít. Robert C. Martin ve své knize „Clean code handbook“ v kapitole o testování zmiňuje důležitost mít stejně kvalitně napsané testy jako zbytek kódu. [4] Varuje před riziky, jež vznikají zanedbáním práce na kvalitních testech a mohou v pozdější fázi vývoje způsobit zvýšení časové

náročnosti na vývoj. Vzniká tak vývojářský dluh, který může být i v konečném důsledku pro vývoj aplikace fatální, jelikož vývojářský tým není schopný dodávat nové funkcionality dostatečně rychle. Tento nadměrný časový tlak může vést k opuštění testovacího prostředí, protože úprava starších testů zabírá příliš mnoho času. V takovém případě se pak značně zvyšuje riziko chyb v produkčním kódu.

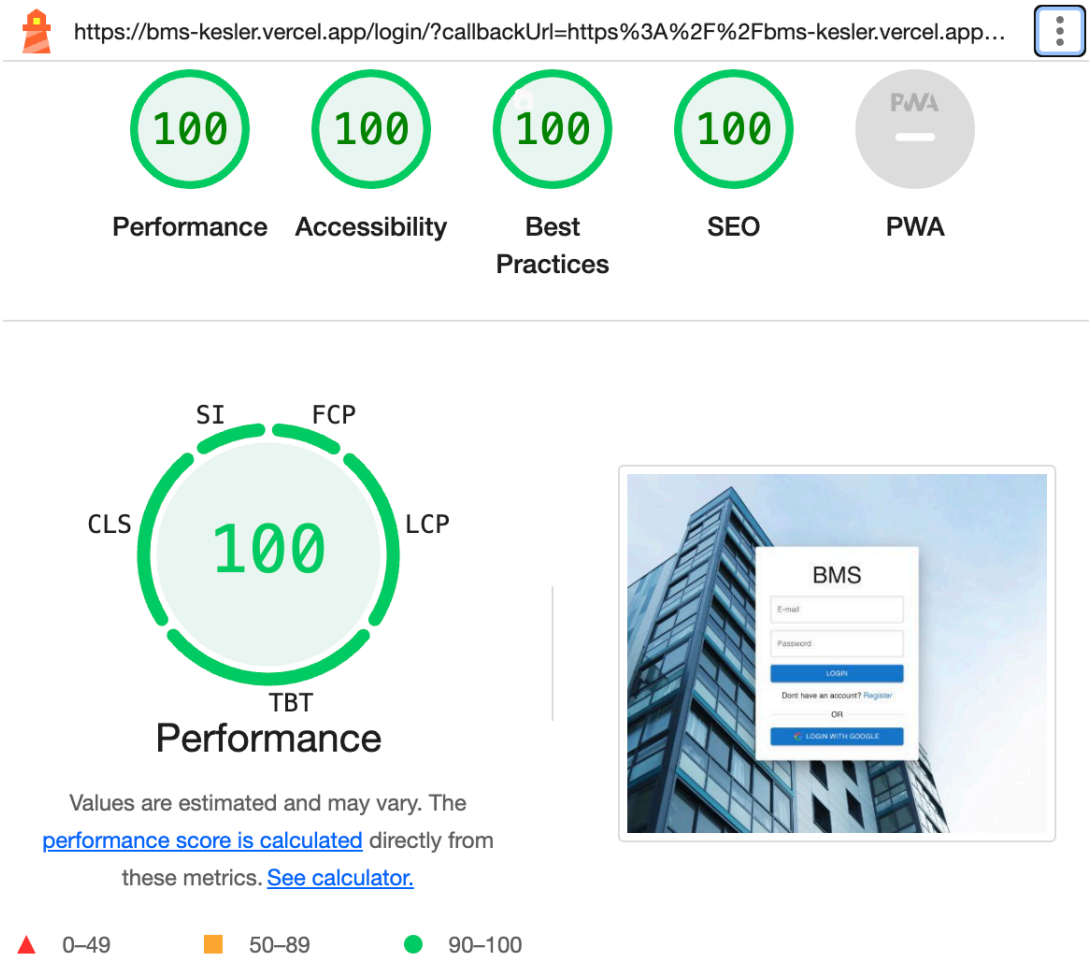
Kvalitní testy by měly vývoj usnadnit, přesto se ale lze setkat s názory, že psaní testů je zbytečně časově náročně. Ve světě, kdy se všechno rapidně vyvíjí, je potřeba se zamyslet co a do jaké míry testovat, aniž by to příliš nezpomalilo vývoj. Nové funkcionality mohou zanikat stejně rychle jako vznikat a je otázkou, zdali je hned testovat či nikoliv. V případě vývoje uživatelského rozhraní můžeme za vhodný test považovat například testování, zda jsou všechna tlačítka dostupná a jestli nejsou na malých zařízeních překryta jiným prvkem rozhraní. Jako nadbytečný test můžeme v některých případech považovat test, který ověřuje, že rozhraní odpovídá na pixel přesně návrhu od designera. První příklad je důležitý pro korektní funkčnost rozhraní, ale druhý už nikoliv. V druhém případě můžeme říct, že při rychlých změnách v uživatelském rozhraní by mohl být zpomalen vývoj na úkor testování. Toto tvrzení ale není obecně platné, vždy to závisí na konkrétním typu projektu a také v jaké fázi se vývoj nachází. To, co je důležité při tvorbě ukázkového produktu, nemusí být nutně prioritou při vývoji aplikace s již ustálenou uživatelskou základnou.

Samotné testy můžeme rozdělit podle několika kritérií například podle toho, zda jsou automatické nebo je nutné je provádět ručně. Dalším způsobem rozdělení může být na nonfunkcionální a funkcionální testy. Nonfunkcionální testy se zaměřují například na rychlost nebo přístupnost aplikace. Funkcionální testy zase ověřují, zda se aplikace chová přesně podle návrhu.

Funkcionální testy dále dělíme do třech kategorií podle toho, s jakou mírou granularity testujeme: jednotkové testy, integrační testy a end to end testy. Jednotkové testy byly zmíněny již v popisu fungování TDD. Jedná se o testy, které jsou nejmenší a testují jednotlivé části kódu, takzvané jednotky. Není dána pevná velikost jednotky, ale většinou se za jednotku považuje funkce nebo třída. Integrační testy jsou testy, které ověřují, zdali nově vytvořené funkcionality či změny fungují dohromady se stávajícím kódem. Do této skupiny testů můžeme zařadit například testy API. Poslední skupinou je end to end testování neboli česky testování od konce ke konci. Je to druh testů, kdy se aplikace testuje jako hotový produkt. Z tohoto důvodu se zpravidla zařazuje ke konci vývoje, kdy je většina funkcionalit hotová.

Jedním z nástrojů pro provádění nonfunkcionálních testů na webech je aplikace Lighthouse, která je součástí prohlížeče Google Chrome. Lighthouse měří 5 různých metrik: výkon, přístupnost, best practice neboli zda jsou používány osvědčené postupy, optimalizaci pro vyhledávače a PWA, což je metrika měřící, do jaké míry se jedná o progresivní webovou aplikaci. Poslední metrika nebude v budoucích verzích Google Chrome nadále podporována. Část protokolu o měření je zobrazena na obrázku číslo 2.4. Každá

metrika může nabývat hodnoty v rozsahu od 0 do 100. Lighthouse označuje červenou barvou výsledky, které jsou neuspokojivé, oranžovou barvou ty, které potřebují zlepšení, a zelenou barvou metriky označuje jako uspokojivé. Součástí hodnocení jsou i rady, jak lépe optimalizovat aplikaci.

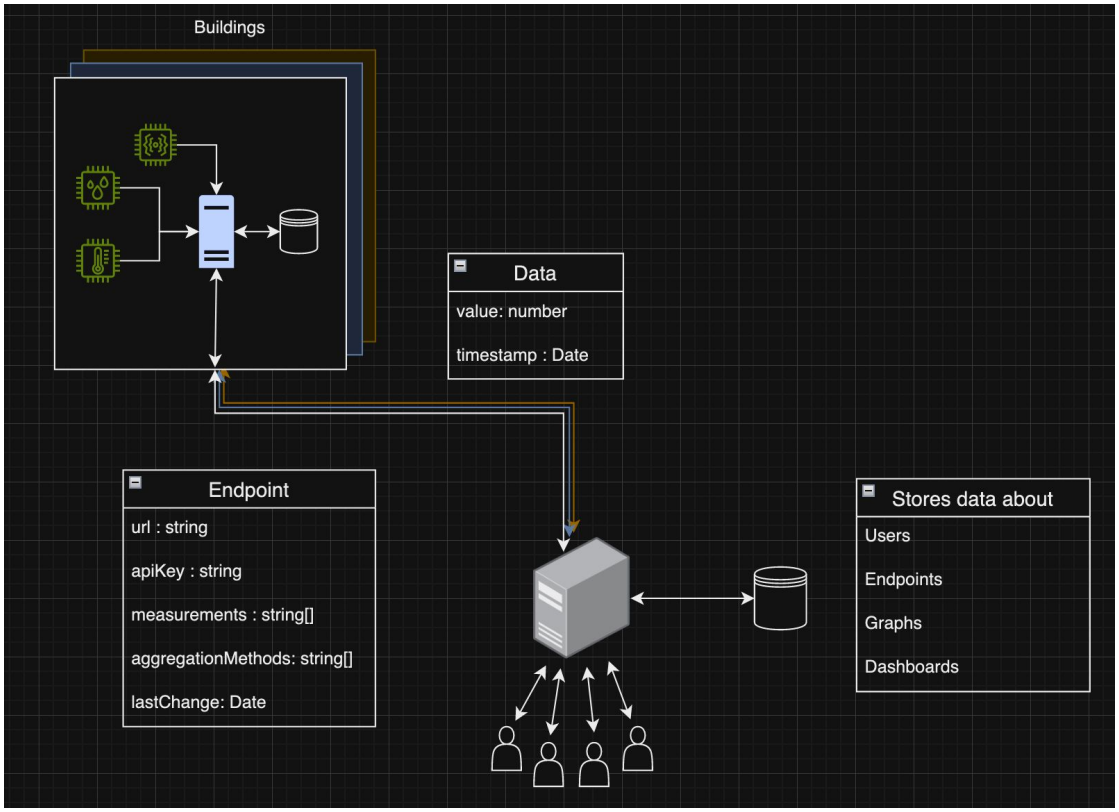


Obrázek 2.4: Lighthouse metriky

3 REALIZACE

3.1 STRUKTURA ŘEŠENÍ

Řešení se skládá ze dvou částí. Schéma návrhu řešení je možné vidět na obrázku číslo 3.1.



Obrázek 3.1: Schéma struktury projektu

První a menší část, která je na obrázku vlevo nahoře, zastupuje jednotlivé budovy. Místní server budovy zpracovává data ze senzorů. Data jsou ukládána a zpřístupněna pomocí API. Přístup k datům je zabezpečený klíčem. Tato část řešení se také stará i o agregaci dat. Agregace dat byla přesunuta na jednotlivé servery budov, jelikož se sníží celkové datové toky proudící internetem. Hlavním důvodem tohoto rozhodnutí bylo zvýšit efektivitu komunikace a snížit výpočetní náročnost a tím i finanční náklady spojené s provozem

hlavní části pro vizualizaci dat. Samotná data jsou pak reprezentována jako časové řady, tedy každá hodnota má i údaj, kdy byla změřena.

Druhá část je webová aplikace pro vizualizaci těchto dat. K tomuto serveru přistupují samotní uživatelé. Budovy jsou pak reprezentovány jako takzvané endpointy neboli česky koncové body, které může uživatel přidávat. Uživatel vyplní pouze adresu a klíč potřebný pro připojení a konfigurační data jsou získána od serveru budovy při prvotní komunikaci. Konfigurační data obsahují údaje o agregačních metodách, dostupných měřeních a poslední změně.

Údaj o poslední změně slouží pro ověření, zdali neproběhla nějaká aktualizace konfigurace od prvotního přidání či poslední aktualizace. Tento údaj je porovnáván při komunikaci s údajem uloženým v databázi a pokud se liší, tak jsou údaje o koncovém bodu aktualizovány. Tento mechanismus slouží k aktualizaci informací o budově, když je systém budovy rozšířen o další senzory. Na základě nových konfiguračních dat ze serveru budovy jsou data příslušných senzorů zpřístupněna automaticky i ve webové aplikaci pro vizualizaci, bez nutnosti uživatele jakkoliv konfigurační data ručně aktualizovat.

Uživatel může dále vytvářet dashboardy a přidávat do nich grafy. Tyto dashboardy jsou v podstatě skupiny grafů sdružené do logických celků. Jako příklad může být třeba dashboard ukazující údaje ze senzorů za uplynulý týden.

3.2 POUŽITÉ NÁSTROJE

Na základě provedené rešerše ohledně dostupných technologií pro vývoj moderních grafických webových aplikací bylo rozhodnuto použít jako hlavní nástroj pro vývoj uživatelského rozhraní knihovnu React. Kromě jeho popularity byl také jedním z rozhodovacích faktorů fakt, že s ním mám z hlediska vývoje největší zkušenosti. Použití Angularu nebo Vue nepřinášelo výrazné benefity oproti Reactu. Jelikož je ale React pouze knihovnou, a nikoliv frameworkem, bylo nadále potřeba vybrat nástroje například pro sestavování či balíčkování kódu.

Řešením tohoto problému je použití frameworku Next.js [24]. Next.js je fullstack framework postavený na Reactu. Kromě nástrojů pro vývoj s sebou tento framework přináší například možnost vytvářet serverové komponenty. Tyto komponenty kromě zlepšení optimalizace pro vyhledávače mají dopad také na zvýšení celkového výkonu a bezpečnosti aplikace.

3.2.1 Routování

Next.js přináší i pokročilé možnosti routování, které se liší od základního přístupu v Reactu. Zatímco v Reactu je třeba explicitně definovat routování pomocí komponent <Route>, v Next.js je využita struktura složek a souborů

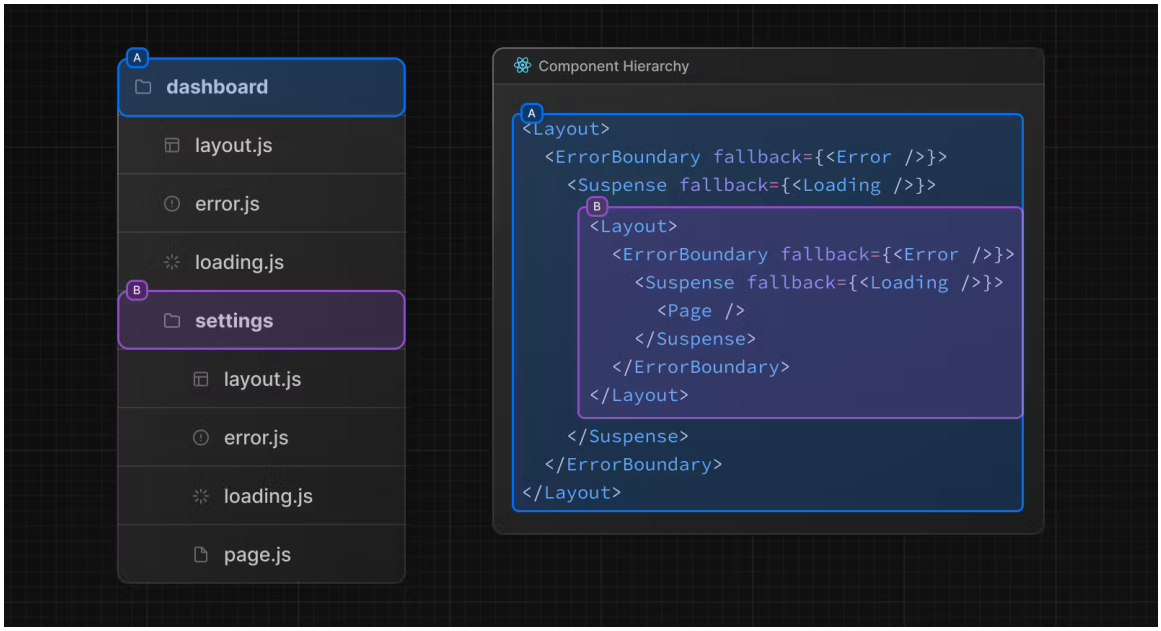
k automatickému routování. V současné verzi Next.js podporuje dva přístupy k routování.

V dřívějších verzích byl používán tzv. „pages router“, kde kořenový adresář nese název „pages“. Od verze 13 je k dispozici modernější „app router“, který má kořenový adresář pojmenovaný „app“. Tyto dva přístupy se neliší jenom názvem kořenového adresáře, ale také funkcionalitou.

Hlavním rozdíl „app routeru“ oproti „pages routeru“ je možnost použít serverové komponenty. „Pages router“ tuto funkcionalitu nenabízí. Dalším rozdílem je, že struktura výsledného URL v „app routeru“ je reprezentována pouze složkami, zatímco v „pages routeru“ je to kombinace názvu složek a souborů.

V obou případech lze vytvářet dynamické cesty a používat layouty, což jsou znovu použitelné komponenty, které jsou součástí více stránek. V rámci layoutů je vhodné implementovat například navigační panel nebo třeba patičku stránky. Hlavní výhodou použití layoutů namísto běžných komponent je, že při přechodu mezi stránkami využívající stejný layout nemusí prohlížeč tuto část znovu vykreslovat.

V „app routeru“ je koncept layoutů rozšířen o další komponenty, jako jsou „loading boundaries“ pro zpracování načítání nebo „error boundaries“ pro zpracování chyb. Tyto komponenty lze podobně jako layouty do sebe vnořovat. Layouty a zmíněné další prvky nemusí být použity vůbec s výjimkou kořenového layoutu projektu, který musí být definován. V rámci každé složky jsou vytvořeny soubory reprezentující jednotlivé komponenty. Názvy těchto souborů jsou předem dané. Struktura a hierarchie uspořádání komponent je zobrazená na obrázku číslo 3.2.



Obrázek 3.2: Struktura routování v app routeru [34]

V levé části obrázku je zobrazena struktura složek a souborů reprezentující obsah „app“ adresáře. Každý soubor ve struktuře definuje jinou komponentu. Hlavním souborem je pak „page.js“, ve kterém je definován samotný obsah stránky. Bez tohoto souboru není pro tuto cestu nic renderováno a cesta je neaktivní. Relativní adresa fialově zvýrazněné stránky by byla „/dashboard/settings“. Samotná adresa je tedy složena z názvu hlavní složky a podsložky. Jelikož pro tuto cestu existuje soubor „pages.js“ je tato cesta aktivní. To samé ale neplatí pro cestu „/dashboard/“ pro kterou bude zobrazena stránka 404 - která značí neexistenci dané stránky. Na pravé straně je pak naznačeno, jakým způsobem jsou do sebe jednotlivé komponenty v rámci hierarchie vnořovány.

App router umožňuje vytvářet flexibilnější routování oproti samotnému Reactu nebo „pages routeru“, ale to vše za cenu vyšší komplexity. V případě nových projektů vytvářených v Next.js je doporučené používat „app router“, zejména kvůli možnosti použití serverových komponent. Z těchto důvodu byl v projektu použit „app router“.

3.2.2 Renderování

Renderování v Reactu je proces vytváření uživatelského rozhraní na základě aktuálního stavu a vstupů do komponenty. V Reactu probíhá renderování v klientské části, tedy v prohlížeči. React ke svému fungování potřebuje JavaScript, bez něho se totiž uživateli zobrazí pouze prázdná stránka, jelikož React vytváří HTML strukturu až za běhu aplikace.

Next.js oproti Reactu nabízí více možností renderování. V základním nastavení jsou všechny stránky předrenderované na serveru. Existují dva typy předrenderování - statické generování (Static generation) a renderování na straně serveru (Server-side rendering)

Statické generování vytvoří HTML strukturu už při sestavování aplikace a vygenerovaná HTML struktura se přepoužívá při každém požadavku na zobrazení stránky. Oproti tomu renderování na straně serveru generuje na serveru pokaždé novou strukturu HTML pro každý jednotlivý požadavek. Tyto metody je možné napříč aplikací kombinovat a vytvářet tak hybridní aplikaci, která využívá pro různé podstránky různé metody renderování. Hlavní výhodou statického generování je rychlost zobrazení stránky, jelikož obsah může být uložen v cache na CDN (content delibery network). Při načtení předrenderované HTML struktury je stále nutné propojit tuto strukturu s virtuálním DOMem, který používá React, aby bylo možné zpřístupnit uživateli interaktivní části. Tomuto procesu se říká hydratace a je prováděn automaticky.

Při sestavování aplikace je možné ověřit pro jakou stránku byla jaká konkrétní metoda použita. Na obrázku číslo 3.3 je zobrazen výstup do konzole z procesu sestavování aplikace. Je zde vidět, že všechny cesty API používají renderování na straně serveru, jelikož pracují s daty, která se mění například v rámci jednotlivých uživatelů. Oproti tomu stránky s formuláři pro vytváření

dashboardů (cesta /dashboard) či stránka pro přidávání jednotlivých koncových bodů (cesta /ednpoint) se nemění a bylo tak možné použít statické generování. Na obrázku je také možné nalézt velikosti JavaScriptu potřebného pro první načtení stránky a pak velikosti přeneseného JavaScriptu při přechodu na z jiné stránky webu. Všechny API cesty mají obě velikosti 0B jelikož negenerují žádné HTML a existují pouze na straně serveru.

```
Route (app)                                 Size     First Load JS
┌ ○ /                                       2.31 kB         113 kB
├ ○ /_not-found                             876 B            83 kB
├ λ /api/auth/[...nextauth]                 0 B               0 B
├ λ /api/dashboards                         0 B               0 B
├ λ /api/dashboards/dashboard               0 B               0 B
├ λ /api/dashboards/dashboard/[dashboardId] 0 B               0 B
├ λ /api/endpoints                          0 B               0 B
├ λ /api/endpoints/endpoint                 0 B               0 B
├ λ /api/endpoints/endpoint/[endpointId]    0 B               0 B
├ λ /api/graphs                             0 B               0 B
├ λ /api/graphs/graph                       0 B               0 B
├ λ /api/graphs/graph/[graphId]             0 B               0 B
├ λ /api/layout                             0 B               0 B
├ λ /api/users                              0 B               0 B
├ λ /api/users/user                         0 B               0 B
├ λ /api/workWithEndpoint                   0 B               0 B
├ ○ /dashboard                              2.79 kB         164 kB
├ λ /dashboard/[dashboardId]                112 kB          276 kB
├ λ /dashboard/[dashboardId]/[graphId]      1.46 kB         175 kB
├ λ /dashboard/[dashboardId]/graph          489 B           167 kB
├ ○ /endpoint                               460 B           169 kB
├ λ /endpoint/[endpointId]                  2.1 kB          176 kB
├ ○ /graph                                  489 B           167 kB
├ ○ /login                                  2.79 kB         176 kB
└ ○ /register                               3.75 kB         177 kB
+ First Load JS shared by all              82.1 kB
  ├ chunks/938-e189aeec1412ea1b.js          26.8 kB
  ├ chunks/fd9d1056-46c0d7c75e49d1f4.js     53.3 kB
  ├ chunks/main-app-e05f32249178e74c.js     226 B
  └ chunks/webpack-50037bb552b3685e.js      1.79 kB


ƒ Middleware                                74.6 kB

○  (Static)   prerendered as static content
λ  (Dynamic)  server-rendered on demand using Node.js
```

Obrázek 3.3: Výstup do konzole při sestavování aplikace

Pro některé části stránky může být nutné použít renderování až na straně klienta. Typickým příkladem je získávání obsahu pomocí React hooku useEffect. Výchozí stav stránky je pořád předrenderován, ale data jsou získána až

na straně klienta. V tomto případě může mít komponenta jako výchozí stav například indikátor načítání, který je později nahrazen získanými daty.

Při použití předrenderování na serveru tedy odpadá problém se zobrazením prázdné stránky, pokud má uživatel vypnutý JavaScript. To je zejména důležité v rámci optimalizace stránky pro vyhledávače neboli search engine optimization zkráceně SEO. Stránky na internetu jsou hodnoceny na základě průzkumu webu boty. Tito boti jsou také označováni jako crawleři. Crawleři mohou mít omezený čas na běh JavaScriptu nebo ho nemusí mít povolený vůbec a stránka pak může být hodnocena hůře a zobrazována ve výsledcích vyhledávání níže. To může vést až k extrémnímu případu, že uživatelé nemohou najít webovou stránku pomocí webových vyhledávačů, jako je například Google nebo Bing.

3.2.3 Nasazení

Při nasazování aplikace bylo rozhodováno mezi dvěma možnostmi. První možností je provozovat aplikaci v kontejneru pomocí Dockeru a nasadit ji například na platformy jako Azure nebo Amazon Web Services (AWS). Druhou možností je využít platformu Vercel, vyvinutou přímo vývojáři Next.js.

Vercel nabízí uživatelsky přívětivé rozhraní pro snadné a rychlé nasazení aplikací. Díky automatickému škálování se aplikace přizpůsobí zvýšenému provozu bez ručního zásahu. Obsahuje i integraci s GitHubem, což usnadňuje automatické nasazování nových verzí. Vercel nabízí i bezplatný plán pro menší projekty.

Nasazení pomocí Dockeru a cloudových platforem poskytuje větší flexibilitu a kontrolu nad infrastrukturou, ale jako platforma pro provoz aplikací byl nakonec zvolen Vercel zejména díky své jednoduchosti a optimalizaci pro Next.js.

3.2.4 Back-end a databáze

Při výběru backendové části bylo vybráno řešení, které používá JavaScript, aby obě části aplikace využívali stejný programovací jazyk. Na základě rešerše a ostatních použitých nástrojů byl vybrán Next.js. Hlavní výhodou Next.js je zaujatý přístup k vývoji, který díky pevně daným pravidlům urychluje vývoj. Pro hlavní aplikaci toto rozhodnutí, dává smysl zejména kvůli použití Next.js pro vytváření frontendové části. Pro vývoj aplikace pro budovy byly použity stejné nástroje, zejména kvůli urychlení vývoje a snížení náročnosti na údržbu kódu.

Při výběru databáze bylo hledáno řešení, které lze provozovat v cloudu a dobře škálovat, tak aby doplnilo řešení provozované na platformě Vercel. Bylo vybráno MongoDB, které lze provozovat na platformě Mongo Atlas. Pro práci s touto databází, byla využita knihovna Mongoose, což je knihovna pro objektové modelování. Umožňuje vytvářet schémata, která definují, jak jed-

notlivé objekty vypadají. Tato knihovna se stará nejen jak datový model vypadá ale například i o validaci dat či komunikaci se samotnou databází.

Jelikož se nejedná o monolitickou strukturu a databázová část je oddělena od aplikační, lze obě části škálovat nezávisle na sobě. Vercel umožňuje spouštění serverových funkcí v různých běhových prostředích. Hlavními jsou klasické Node.js a Edge běhové prostředí. Edge se snaží o co nejmenší odezvu, a proto se kód vykonává co nejblíže uživateli, na rozdíl od Node.js běhového prostředí, které je vázáno na regionální infrastrukturu. Edge běhové prostředí implementuje pouze některá Node.js API. Z tohoto důvodu je provoz v Edge běhovém prostředí méně výpočetně, a tedy i finančně, náročný.

Přestože běhové prostředí Edge by bylo vhodnější pro komunikaci s databází, knihovna Mongoose v současnosti nepodporuje toto běhové prostředí. Pro komunikaci s databází její ovladač používá „net API“, které Vercel v momentální verzi Edge běhového prostředí nepodporuje. Z tohoto důvodu bylo použito Node.js běhové prostředí.

3.2.5 Vizuální část

Pro stavbu vizuální stránky bylo na základě rešerše rozhodnuto použít již existující design systém. Hlavním důvodem výběru použití již existujícího design systému místo vytváření vlastních komponent bylo zrychlení vývoje a usnadnění budoucích úprav díky použití témat.

Ze zmíněných design systémů byl vybrán Material Design se svojí implementací Material UI. Tento design systém byl zvolen, zejména kvůli detailní dokumentaci, která oproti konkurenčním design systémům klade důraz na přístupnost.

3.3 HLAVNÍ APLIKACE

Hlavní část práce se stará o vizualizaci dat z budov. Aplikace uživatelům umožňuje vytvořit si účet, účtu přiřadit jednotlivé koncové body budov a na základě dat z těchto koncových bodů pak vytvářet dashboardy neboli přehledy vytvořené z tabulek a grafů.

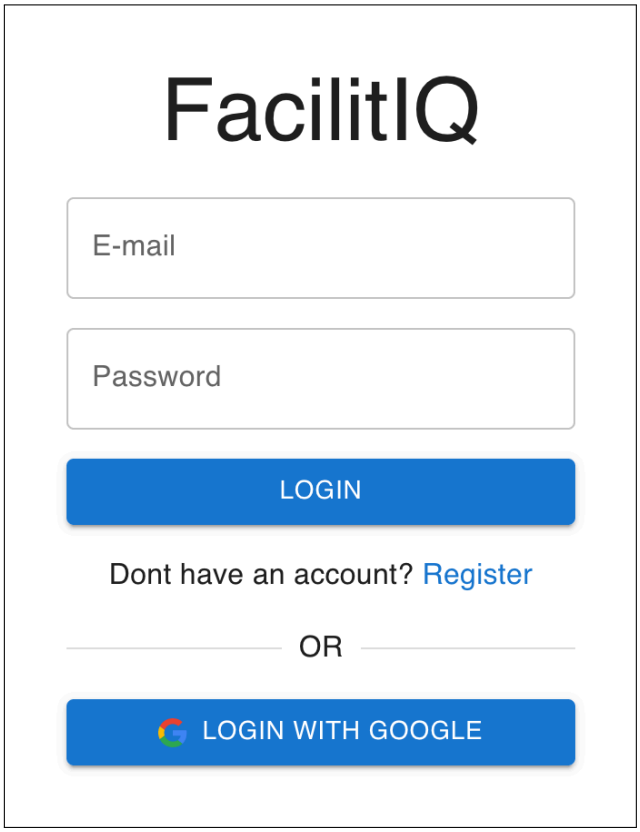
3.3.1 Uživatelské účty

Aby bylo zamezeno neoprávněnému přístupu k prostředkům, bylo v aplikaci vytvořeno rozhraní pro přihlašování a vytváření uživatelských účtů. Nepřihlášený uživatel má přístup právě k těmto dvěma podstránkám, tedy přihlašovací stránce a registrační stránce.

Přihlašovací stránka obsahuje formulář, který je na obrázku číslo 3.4. Přihlašovací formulář je umístěný na středu stránky a obsahuje pole pro zadání hesla a emailové adresy, která slouží k jednoznačné identifikaci uživatele.

Na této stránce je i odkaz na registrační stránku a také možnost přihlášení pomocí účtu Google.

Přihlašování pomocí účtu Google používá standard OAuth2 [20]. Jako implementace tohoto standardu byla použita knihovna NextAuth. Bylo nutné vytvořit si účet Google a na stránkách Google cloud console vytvořit projekt. V tomto projektu byly vyplněny definované údaje o projektu: typ aplikace, URL, ze kterého se dotazuje server a ze kterého se dotazuje webová aplikace. Následně byly vygenerovány přístupové údaje, které jsou použity při komunikaci s Google API.



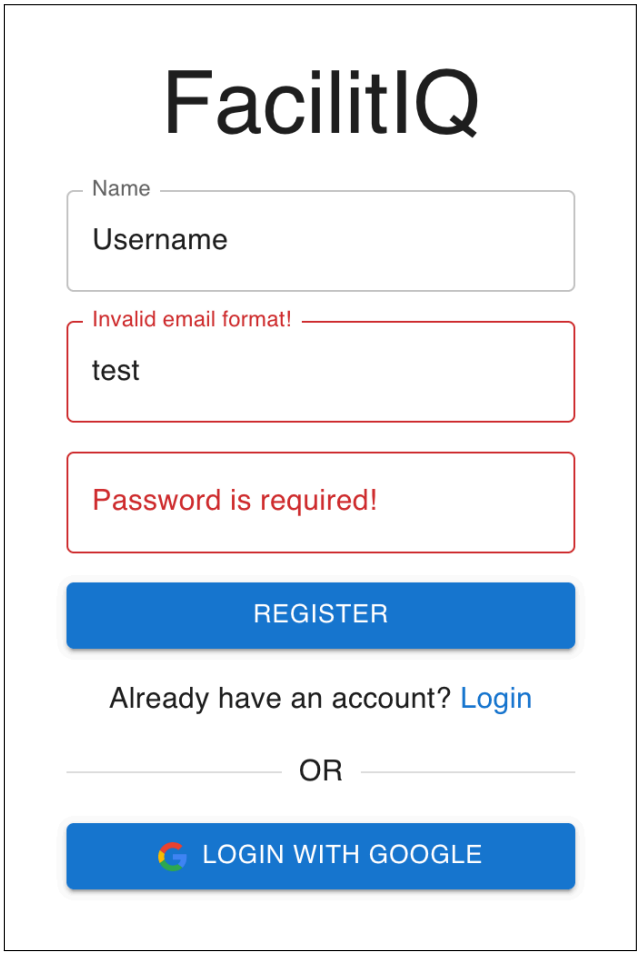
Obrázek 3.4: Přihlašovací formulář

Pro použití v produkčním režimu bylo nutné zažádat o ověření aplikace. Bez tohoto ověření je aplikace provozována z hlediska Google v testovacím režimu a je možné se přihlašovat pouze z účtů, které byly v rámci tohoto projektu na Google cloud console autorizovány pro testování. Těchto testovacích účtů je možné přidat až sto. Tato API v základním nastavení umožňuje vytvořit až 10 000 nových tokenů denně. Pro navýšení limitu je nutné kontaktovat podporu. Pro testování a pilotní provoz aplikace je tento limit plně dostačující a nebylo nutné ho navyšovat.

Při přihlašovaní je z databáze získán uživatelský účet a uložen do uživatelského kontextu. Kontext je způsob, jak spravovat stav aplikace globálně. Díky kontextu je možné přistupovat ke sdíleným položkám bez nutnosti

je předávat pomocí vstupů do komponent. V rámci implementace uživatelského kontextu byly vytvořeny funkce pro automatické získání dashboardů a koncových bodů z databáze na základě přihlášeného uživatele. K těmto datům přistupuje například navigační lišta. Po získání uživatelského účtu je uživatel přesměrován na úvodní stránku aplikace s relativní adresou „/“, která slouží jako úvodní přivítání.

Druhou zmíněnou webovou stránkou dostupnou nepřihlášeným uživatelům je registrační stránka. V pozadí této stránky je obrázek budovy a na jejím středu je registrační formulář. Tento formulář je na obrázku číslo 3.5. Jsou zde vidět všechny tři možné stavy formuláře - akceptovaný vstup, vstup s chybou a prázdný vstup. Pouze email může mít vstup s chybou, protože jako jediný je validován pomocí regulárního výrazu. Jméno, email a heslo jsou povinné položky pro vytvoření uživatele.



Obrázek 3.5: Registrační formulář

Emailová adresa slouží k unikátní identifikaci uživatele a každý email může být proto použit pouze jednou. V případě prvotního přihlášení pomocí Google účtu, je také vytvořen nový uživatelský účet. Tento účet je vytvořen pouze v případě, že v databázi neexistuje účet pro tuto emailovou adresu. Takto

vytvořený účet nemá nastavené žádné heslo, a proto v současné verzi aplikace je pro přihlašovaní k tomuto účtu vždy nutné použít přihlášení pomocí služby Google. Pokud byl pro danou emailovou adresu vytvořen nejprve účet s heslem, lze použít obě možnosti přihlášení.

Po kliknutí na tlačítko registrovat je odeslán POST požadavek s údaji o uživateli na relativní adresu „/api/users“. Namísto textu register v tlačítku je zobrazen rotující kruh signalizující komunikaci s API. Tlačítko je v průběhu komunikace ve zakázaném stavu (disabled) a není možné na něj znovu kliknout a tudíž uživatel nemůže odeslat najednou více než jeden požadavek na server. Při úspěšném vytvoření uživatelského profilu, je uživatel přesměrován na přihlašovací stránku. V případě chyby je v pravém dolním rohu zobrazena hláška s chybou a tlačítko registrovat se vrací do původního stavu. V současné verzi jsou implementovány pouze dvě chybové hlášky. První hláškou je, že uživatelské konto pro tuto emailovou adresu již existuje a druhá chybová hláška je chyba při registraci. Tato hláška je obecná a je zobrazena pro všechny ostatní chyby, jako například při chybě komunikace s registračním API.

Oba formuláře byly implementovány pomocí kombinace komponent z knihovny Material UI a knihovny react-form-hook, která se stará o funkcionalitu formuláře. Všechny komponenty z MUI jsou renderovány pomocí komponenty „Controller“, která se mimo renderování stará i o validaci vstupů.

Validaci formuláře je možné provozovat v pěti různých režimech v závislosti na tom, kdy je vstup validován. První možností je validovat vstup před odesláním formuláře. Druhou možností je validovat vstup při každé „onChange“ události. To ale znamená, že například při zadání prvního znaku při vyplňování emailu je vstup označen jako chybný, dokud není dopsán validní zbytek emailové adresy, což může být pro uživatele matoucí. Lepší alternativou je třetí možnost, a to validovat vstup při každé změně vstupu až po první „onBlur“ události. Tato událost je spouštěna pokaždé, když uživatel opustí zadávací pole ať už kliknutím jinam na obrazovce nebo třeba stisknutím tlačítka TAB na klávesnici. Obě z těchto možností mohou mít podle dokumentace [21] negativní vliv na výkon, jelikož jsou vázány na „onChange“ událost. Čtvrtou možností je validace na oba typy událostí najednou, což sebou přináší stejný problém jako má možnost číslo dvě, tedy matoucí zobrazení že vstup je chybný při prvotním zadávání. Poslední možností je validovat vstupní pole při „onBlur“ události. Tento přístup je více uživatelsky přívětivý než validace až před odesláním formuláře a navíc nemá negativní dopad na výkon. Z tohoto důvodu byl tento režim použit na všech formulářích napříč aplikací.

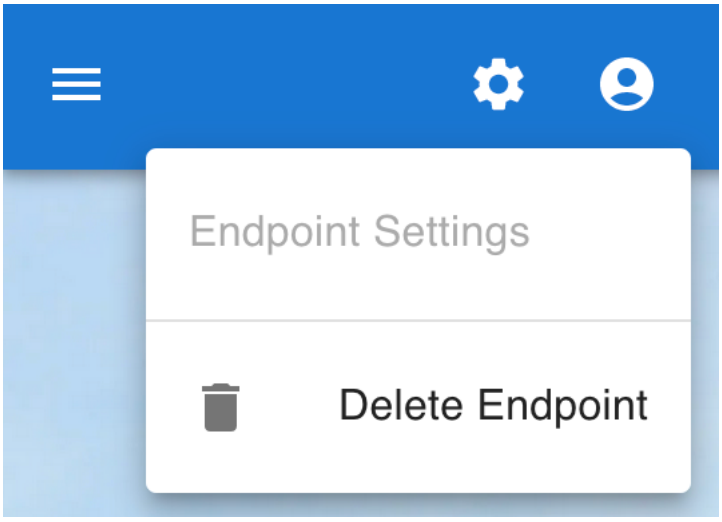
3.3.2 Navigační rozhraní

Navigační rozhraní se skládá ze dvou částí: boční lišty, která je vytvořena pomocí MUI komponenty „drawer“, a horní lišty, která je vytvořená pomocí

komponenty „appbar“. V dokumentaci Material Designu [13] je doporučeno používat pouze jeden navigační prvek, a proto horní lišta slouží pouze jako doplňková, zatímco hlavní navigaci obstarává boční lišta. Obě lišty mají nastavenou fixní pozici a jsou tedy stále na obrazovce i při scrollování dokumentu.

Z důvodu optimalizace rychlosti přechodu mezi jednotlivými stránkami je navigační rozhraní součástí layoutu, který je společný pro všechny stránky dostupné po přihlášení. Tento layout kromě horní a boční lišty obsahuje i obrázek budovy [36], který vyplňuje zbytek obrazovky. Tento obrázek má fixní pozici a zIndex nastavený na hodnotu -1, aby byl vždy za ostatními komponentami.

Horní lišta se nachází na obrázku číslo 3.6. V levé části obsahuje tlačítko pro otevření boční lišty, a na pravé straně jsou umístěna pomocná tlačítka. Tlačítko profilu je dostupné na každé stránce a obsahuje nabídku se jménem uživatele a tlačítkem pro odhlášení. V budoucích verzích aplikace by se zde měly nacházet také možnosti pro správu uživatelského účtu, jako je změna jména, hesla nebo možnost smazat uživatelský účet.



Obrázek 3.6: Horní navigační lišta

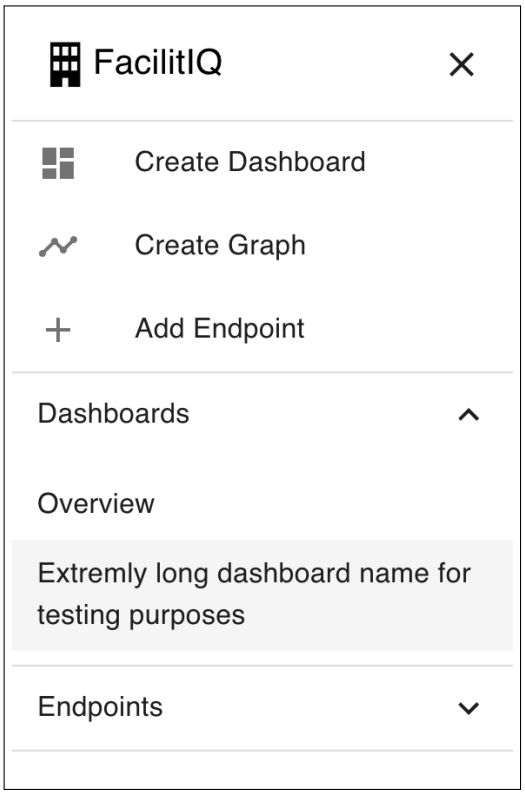
Tlačítko s ikonou ozubeného kola je doplňkové a nenachází se na všech stránkách. Obsahuje možnosti specifické pro danou stránku. Například v případě stránky pro editaci koncového bodu se v menu nachází možnost konkrétní koncový bod smazat. Toto menu je otevřené na obrázku číslo 3.6.

Boční lišta se nachází na obrázku číslo 3.7. Po otevření překrývá stránku včetně horní lišty. Je možné ji zavřít kliknutím mimo boční lištu či na tlačítko v horní části. Samotná boční lišta je rozdělená do čtyř částí pomocí komponenty „Divider“. První část obsahuje, kromě tlačítka pro zavření, logo aplikace, které po kliknutí vede na úvodní stránku aplikace. Další část obsahuje tři tlačítka pro přístup k vytváření dashboardů, grafů a koncových bodů. Poslední dvě sekce jsou rozbalovací menu, které obsahují přístup k již vytvo-

řeným koncovým bodům a dashboardům. Šířka boční lišty je shora omezena na 400px a zdola je omezena hodnotou 200px. Na obrázku je možné vidět, že dlouhé texty, které se nevejdou na jeden řádek, se zalamují. Zalomený text je zároveň ve stavu :hover a slouží jako ilustrace efektu zvýraznění, pokud se nad položkou menu nachází kurzor.

V rámci implementace navigačních komponent byl vytvořen navigační kontext, který vznikl hlavně z důvodu práce s menu reprezentovaným na obrázku číslo 3.6 ozubeným kolem. Toto menu je, jak bylo zmíněno, volitelné a bylo implementováno rozdílně na různých stránkách.

Na každé stránce je možné pomocí kontextu přistupovat k funkci „setRightMenu“, která umožňuje nastavovat React komponentu, která bude zobrazena vedle tlačítka pro správu uživatelských profilů. Je tedy možné místo ozubeného kola použít například tlačítko s jinou ikonou nebo dokonce více tlačítek. Při implementaci tohoto React elementu je důležité myslet na responzivitu. Je vhodné použít tlačítka, která otevírají menu, jako je to v případě správy profilu nebo nastavení koncových bodů. Není vhodné vkládat položky menu přímo do horního panelu kvůli limitovanému místu na horní liště a také kvůli zachování designové celistvosti. Z tohoto důvodu jsou všechna nastavení na jednotlivých stránkách označena ikonou ozubeného kola.



Obrázek 3.7: Boční navigační lišta

Ve zdrojovém kódu číslo 1 se nachází ukázka části implementace kontextu. Konstanta data je deklarována pomocí React hooku useRef, což umožňuje

uchovávat referenci na objekt s daty, jež slouží k interakci mezi navigačním menu a ostatními komponentami. Tímto způsobem uchovaná data jsou zachována i při přererenderování komponent. Změna v těchto datech nevyvolává přerenderování. Naopak stavová proměnná RightMenu vyvolá přerenderování komponent. Tato stavová proměnná je definovaná pomocí React hooku useState.

React hook useEffect je využitý k implementaci reakce na změnu cesty aplikace, získaného pomocí hooku usePathname. Tato funkce je vyvolána při každé změně cesty a slouží k odstranění menu a vyčištění dat, která byla použita na předchozí stránce. Tím je zajištěna korektní aktualizace uživatelského rozhraní.

```
1 const data = useRef<any>({ layout: null, graphs: null });
2 const pathname = usePathname();
3 const [RightMenu, setRightMenu] = useState<React.ComponentType<any> |
     null>(null);
4   useEffect(() => {
5     if (RightMenu) {
6       setRightMenu(null);
7       data.current.layout = null;
8     }
9   }, [pathname]);
```

Zdrojový kód 1: Implementace volitelného menu v kontextu

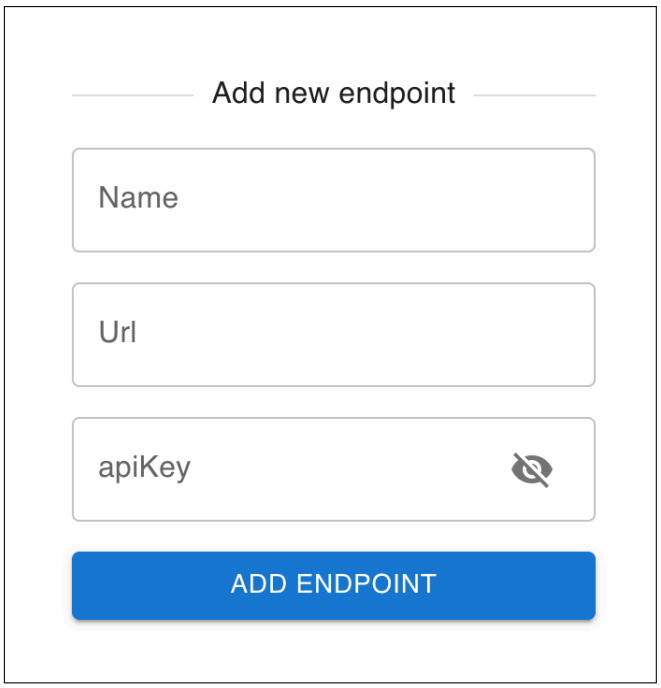
3.3.3 Koncové body

Hlavní část aplikace se zabývá vizualizací dat pomocí grafů a tabulek. Prvním krokem k vytvoření grafu nebo tabulky je ale přidání koncového bodu, který slouží jako zdroj dat pro tyto vizualizace.

Přidávání koncových bodů se provádí na stránce s relativní adresou „/endpoint/“, která je dostupná prostřednictvím navigačního menu. Ve středu stránky se nachází formulář pro přidávání koncových bodů, který je zobrazen na obrázku číslo 3.8. Formulář se skládá ze tří polí. První pole slouží k zadání jména koncového bodu, které bude následně použito například v seznamu koncových bodů v navigační liště. Druhé pole slouží k zadání adresy, která identifikuje koncový bod. Třetí pole je určeno pro zadání klíče, který slouží k přístupu k API.

Z důvodu zvýšení úrovně soukromí je pole pro zadání API klíče typu heslo, což znamená, že zadávané znaky jsou zobrazeny jako tečky. Pro ověření správnosti zadávaného API klíče je součástí formuláře tlačítko, které po kliknutí umožní zobrazit text ve vstupním poli.

Po kliknutí na tlačítko „ADD ENDPOINT“ je ověřeno, že lze navázat spojení se zadanou URL a následně je odeslán POST požadavek na relativní adresu „/api/endpoints/endpoint“. V těle tohoto požadavku jsou definována da-



Obrázek 3.8: Formulář na přidávání koncového bodu

ta z formuláře. Během zpracování tohoto požadavku na serveru je navázáno spojení se serverem budovy a jsou získány informace o koncovém bodu. Následně jsou data z formuláře společně se získanými informacemi o konfiguraci koncového bodu uložena do databáze. Během zpracování požadavku je text tlačítka nahrazen rotujícím kruhem indikujícím zpracování požadavku a tlačítko je v zakázaném stavu a nereaguje na další kliknutí.

V situaci, kdy není možné navázat spojení nebo když se vyskytne chyba při ukládání do databáze, je v pravém dolním rohu aplikace zobrazeno okno s příslušnou chybovou hláškou. V případě úspěšného vytvoření koncového bodu v databázi je v odpovědi od serveru vrácen vytvořený bod. Tento bod je přidán do seznamu bodů v uživatelském kontextu a uživatel je přesměrován na úvodní stránku aplikace.

Vytvořené koncové body jsou zobrazeny v navigační liště. Po kliknutí na libovolný koncový bod je uživatel přesměrován na editační stránku daného koncového bodu. Jedná se o stejný formulář jako je na obrázku číslo 3.8 s tím rozdílem, že namísto textu „ADD ENDPOINT“ je text nahrazen textem „EDIT ENDPOINT“. Jednotlivá vtupní pole jsou předvyplněná uloženými daty.

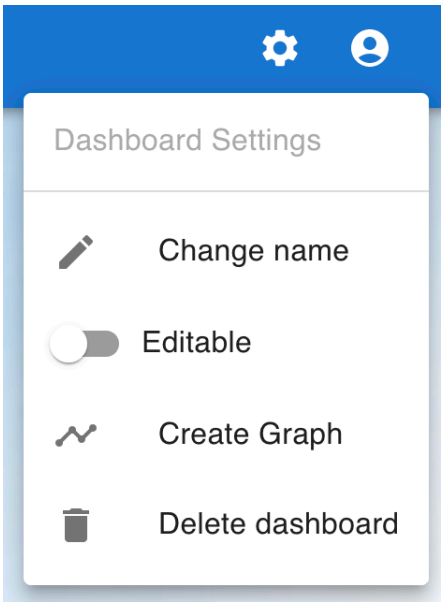
Po kliknutí na tlačko „EDIT ENDPOINT“ je nejprve zobrazena varovná hláška, která uživatele upozorňuje na případné problémy, které může změna uloženého koncového bodu způsobit. Například se mohou rozbít již vytvořené grafy, které používají daný koncový bod. Uživateli je doporučeno spíše vytvořit nový koncový bod. Pokud uživatel přesto souhlasí je odeslán požadavek na server. V případě úspěšné změny obsahuje tělo odpovědi serveru data o změněném koncovém bodu. Na základě těchto dat je upraven se-

znam koncových bodů v uživatelském kontextu a uživatel je přesměrován na úvodní stránku aplikace. Editační stránka obsahuje v horní liště menu, které slouží k odebrání koncového bodu. Toto menu je na obrázku číslo 3.6.

3.3.4 Dashboardy

Dalším krokem po přidání koncového bodu je vytvoření dashboardu, což je prostředek pro prezentaci a organizaci grafů. Dashboardy fungují jako místa, kam lze umístit grafy a seskupit je do logických celků. Dashboardy se vytváří pomocí formuláře. Odkaz na stránku s formulářem lze nalézt v boční navigační liště. Jediným vstupním polem formuláře je jméno. Zadané jméno nemusí být unikátní a je tedy možné vytvořit dashbordy se stejným jménem. Po úspěšném vytvoření dashboardu, je uživatel přesměrován na stránku nově vytvořeného dashboardu.

Na této stránce se v horní liště nachází menu, které obsahuje čtyři možnosti. Toto menu se nachází na obrázku číslo 3.9.

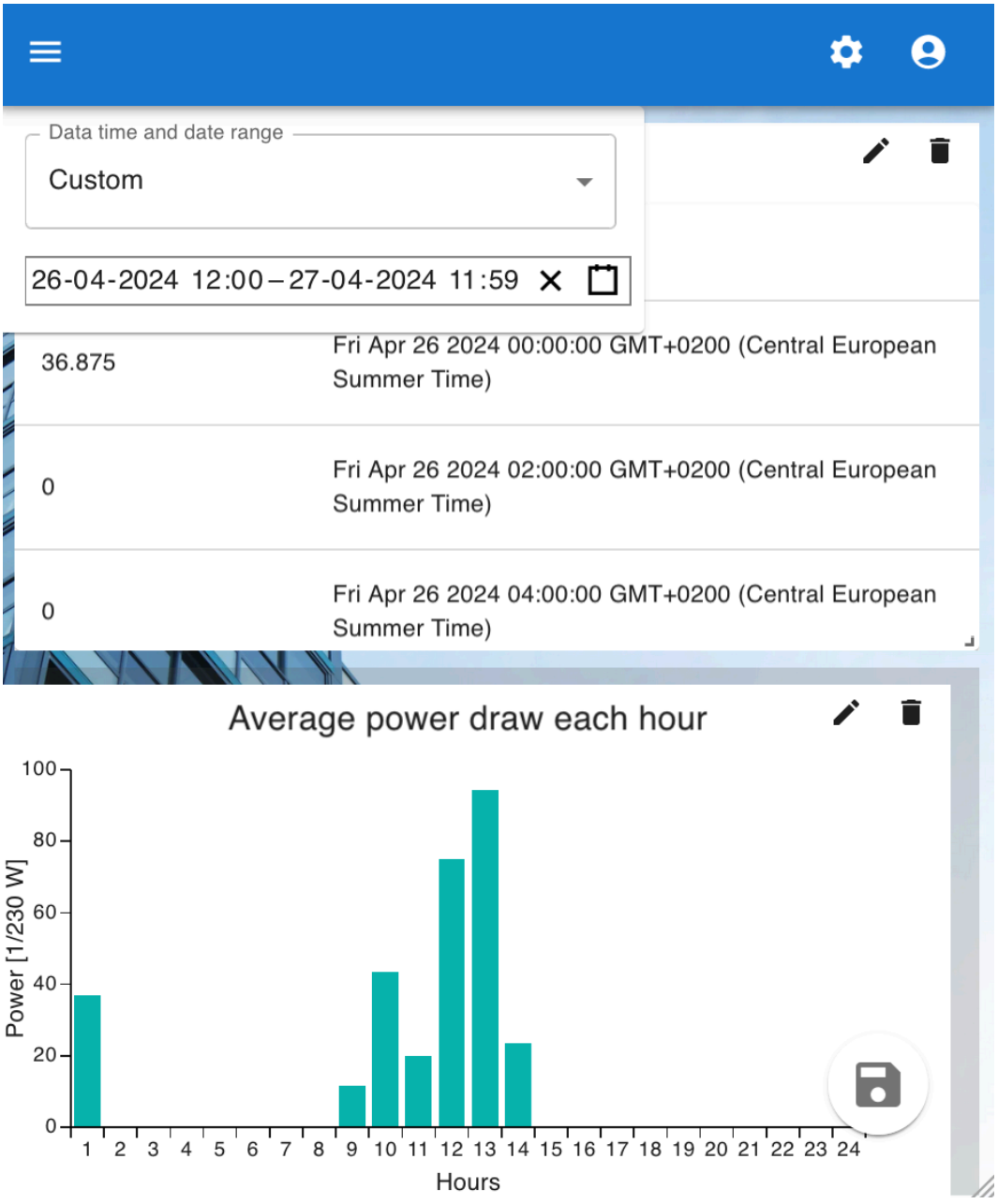


Obrázek 3.9: Menu na stránce dashboardu

První možností je úprava jména dashboardu. Po kliknutí na tlačítko „Change name“ je menu zavřeno a je otevřen dialog. Tento dialog je na středu obrazovky a obsahuje pole s předvyplněným aktuálním jménem dashboardu. Uživatel má možnost jméno změnit a uložit změny nebo tento dialog zavřít kliknutím na tlačítko s nápisem „Cancel“ nebo mimo hlavní okno dialogu. Když je dialog otevřený, je zbytek aplikace lehce ztmavený, aby se pozornost uživatele koncentrovala na nově otevřené okno.

Druhou možností v menu je přepínač editačního režimu. Dashboard se zapnutým editačním režimem je zobrazený na obrázku číslo 3.10. Editační režim umožňuje interaktivně upravovat velikost a pozici jednotlivých grafů

a tabulek. Velikost se upravuje pomocí tažení pravého dolního rohu. Pozici je možné změnit přetažením grafu nebo tabulky na jiné místo v dashboardu. Grafy se automaticky zarovnávají do připravené mřížky, která je pro různé velikosti obrazovky jinak velká. Na obrázku je spodní graf v průběhů změny pozice. Šedou barvou pod grafem je naznačená pozice, kam se v mřížce graf přichytí po skončení tažení.



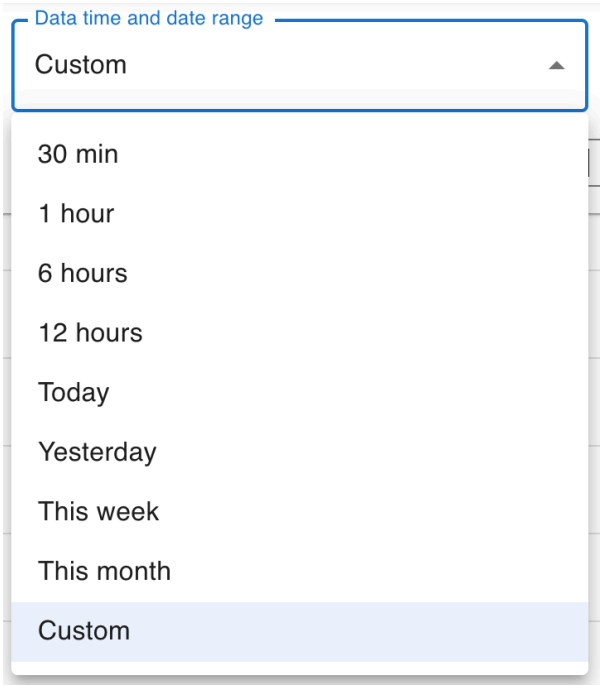
Obrázek 3.10: Editační režim dashboardu

Pro implementaci této funkcionality byla použita knihovna „react-grid-layout“ [37]. Hlavními výhodami této knihovny oproti jiným je, že je kompletně napsaná v Reactu a je plně responzivní. To znamená, že knihovna umož-

ňuje použití breakpointů a automatické rozšíření kontejneru v závislosti na velikosti obrazovky. V dokumentaci této knihovny je také část věnovaná optimalizaci, která byla velmi důležitá pro implementaci spolu s grafy z komponentní knihovny MUI. Bez optimalizace docházelo k překreslování grafů při změně velikosti a pozice, což mělo negativní dopad na výkon. Tuto knihovnu využívají například Grafana nebo AWS CloudFront Dashboards.

Editační režim dále umožňuje přejít na editační stránku grafu pomocí tlačítka se symbolem tužky v pravém horním rohu. Napravo od tohoto tlačítka se nachází tlačítko se symbolem koše, které slouží ke smazání grafu. Po kliknutí na toto tlačítku je vyvolán dialog, který slouží k potvrzení této volby. Tato funkcionalita byla implementována z důvodu zamezení nechtěného smazání grafu.

Další součástí editačního režimu je komponenta umožňující editaci časového rozsahu pro zobrazované grafy a tabulky. Tato komponenta se nachází v levém horním rohu. Otevřené menu je zobrazeno na obrázku číslo 3.11. Uživatel má na výběr z devíti možností. Všechny možnosti s výjimkou volby „Custom“ se upravují v závislosti na konkrétním datu. Například pokud je zvolena možnost „Yesterday“ neboli česky včera, budou zobrazována data vždy z předchozího dne. Oproti tomu možnost „Custom“ umožňuje pomocí vstupního pole zadávat konkretní datum a čas. Tento rozsah je pevně daný a není závislý na konkrétním datu, kdy je dashboard otevřen.



Obrázek 3.11: Možnosti nastavení časového rozmezí

Knihovna MUI obsahuje komponentu pro výběr časového rozsahu pouze ve verzi pro, která je zpoplatněná částkou 15 dolarů za měsíc. Z tohoto důvodu byla místo MUI použita knihovna „React-DateTimeRange-Picker“ [38].

Komponenta vizuálně zapadá do kontextu aplikace a nebylo ji nutné příliš upravovat. Na základě této komponenty byla vytvořena vlastní komponenta „DatePicker“, která je tvořena ze zmíněného vstupu pro výběr rozsahu a rozbalovacího menu. Vstup pro výběr data, je zobrazen pouze při výběru možnosti „Custom“. Dále se tato komponenta stará o převod na zvolené volby na konkrétní časový údaj.

Poslední součástí editačního režimu je tlačítko se symbolem diskety v pravém dolním rohu. Toto tlačítko slouží k uložení změn v layoutu grafů a tabulek a zobrazovaném časovém rozsahu. Změny jsou uloženy i v případě, kdy uživatel vypne editační režim pomocí přepínače, kterým tento editační režim zapnul. Změny nejsou zachovány, pokud je změněna stránka nebo když je stránka obnovena uživatelem.

Třetí možností v menu je tlačítko vytvořit nový graf. Po kliknutí je uživatel přesměrován na stránku s formulářem pro vytváření grafu. Při použití tohoto tlačítka namísto tlačítka v boční navigační liště formulář obsahuje o jedno vstupní pole méně. Neobsahuje rozbalovací menu s výběrem dashboardu. Ten je předán v pozadí a není možné ho tedy znovu vyplňovat.

Poslední tlačítko v menu umožňuje smazání dashboardu. Uživateli je po kliknutí zobrazen dialog, který slouží k potvrzení smazání. V případě smazání je uživatel přesměrován na stránku s formulářem pro vytváření dashboardu. Společně s dashboardem jsou smazány i grafy, které dashboard obsahoval.

3.3.5 Grafy a tabulky

K vytváření grafů slouží formulář, který obsahuje celkem devět vstupních polí. Jak bylo zmíněno v předchozí kapitole, pokud uživatel přejde na vytváření grafů z menu dashboadu, tak tento formulář obsahuje pouze osm polí, jelikož rozbalovací menu s výběrem dashboardu je skryté.

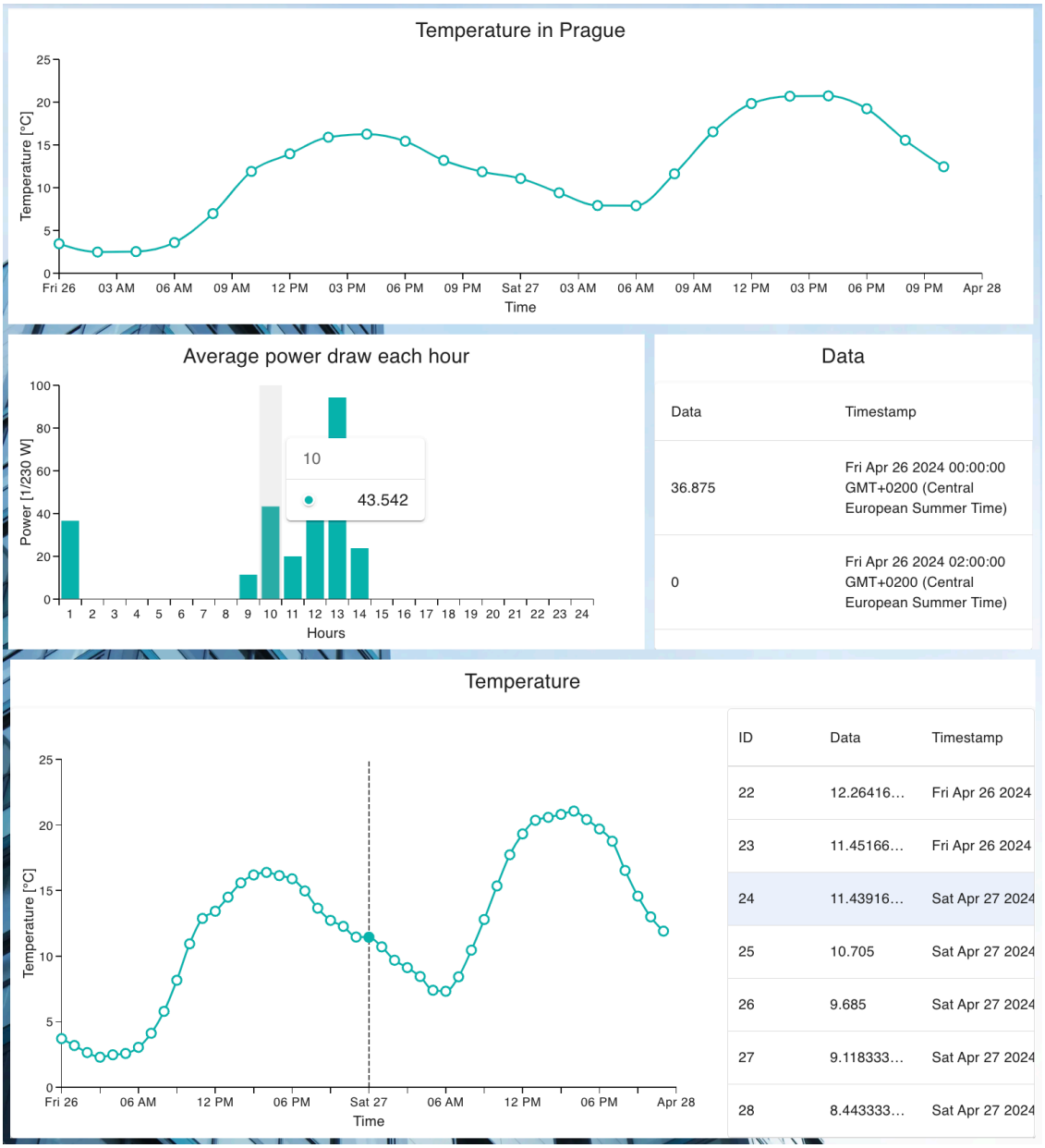
Uživatel může při vytváření grafů vybírat ze čtyř možností vizualizace. K dispozici je spojitý graf, sloupcový graf, tabulka a interaktivní kombinace tabulky se spojnicovým grafem. Jednotlivé vizualizace jsou zobrazeny na obrázku číslo 3.12.

Grafy umožňují interaktivně zobrazit aktuální hodnotu bodu nebo sloupce. Tato funkcionalita je zobrazená na obrázku u sloupcového grafu. V případě připojeného ukazovacího zařízení jako je myš nebo touchpad, stačí kursor umístit nad příslušnou část grafu. Bez připojeného ukazovacího zařízení je tato funkcionalita dostupná po kliknutí do grafu. Další implementovaná interaktivní možnost, je při použití vizualizace kombinující graf s tabulkou. Po kliknutí do grafu je zvýrazněn vybraný bod v tabulce hodnot. Tato funkcionalita se nachází v dolní části obrázku. U každé vizualizace je dále možné zvolit počet bodů, které tabulka nebo graf budou obsahovat.

Formulář obsahuje rozbalovací menu s výběrem koncového bodu. Pokud je zvolen koncový bod, je možné vybrat agregační metodu a zvolit měření z příslušných rozbalovacích menu. Tato menu jsou prázdná, pokud není zvolen žádný koncový bod. Zobrazené možnosti v těchto menu, jsou získána

z příslušné konfigurace koncového bodu, která je uložena v databázi.

Dále je nutné vyplnit jméno grafu, které je použité jako titulek. Formulář obsahuje i dvě volitelná vstupní pole: jméno osy x a jméno osy y.



Obrázek 3.12: Možnosti vizualizace

Po úspěšném vytvoření grafu v databázi, je uživatel přesměrován na stránku zvoleného dashboardu. V případě chyby je stejně jako u ostatních formulářů v pravém dolní rohu obrazovky zobrazena chybová hláška.

Editační stránka grafu obsahuje stejný formulář jako v případě vytváření grafu nového. Data v tomto formuláři jsou předvyplněná aktuálními hodnotami. Editační formulář má jiný nadpis a obsah tlačítka, kde je místo „Create

graph“ napsáno „Edit graph“. Pod formulářem je tlačítko pro návrat k dashboardu bez uložení změn.

3.3.6 Testování

Pro aplikaci byly napsány jednotkové testy, které ověřují funkčnost komunikace s databází. Jako testovací framework byl použit Jest. Na obrázku číslo 3.13 je zobrazeno pokrytí testy. Celkem bylo napsáno 68 testů, které pokrývají 100 % kódu ve složce „services“, tedy ve složce kde jsou soubory obsahující pomocné funkce pro komunikaci s databází. Jsou tedy i otestovány krajní případy, kdy jsou například vstupní data do funkce neplatná. Tyto testy jsou integrovány do CI/CD pipeline, která je blíže popsána v kapitole 3.5.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
services	100	100	100	100	
dashboard.ts	100	100	100	100	
dashboardUser.ts	100	100	100	100	
endpoint.ts	100	100	100	100	
endpointUser.ts	100	100	100	100	
graph.ts	100	100	100	100	
user.ts	100	100	100	100	
services/tests	100	100	100	100	
seedDB.ts	100	100	100	100	

```
Test Suites: 6 passed, 6 total
Tests:       68 passed, 68 total
Snapshots:   0 total
Time:        1.822 s, estimated 2 s
```

Obrázek 3.13: Pokrytí testy hlavní aplikace

Hlavní aplikace, neobsahuje integrační testy. Vytvořené neveřejné API bylo během vývoje testováno ručně pomocí programu Postman.

V rámci end to end testování byla aplikace ručně testována v prohlížečích Safari a Google Chrome ve třech rozlišeních: mobil v rozlišení 375 na 812 pixelů, tablet v rozlišení 810×1080 pixelů a počítač v rozlišení 1920 na 1080 pixelů. Kromě celkové funkčnosti aplikace se end to end testy zaměřily zejména na responzivitu.

Pro nonfukncionální testování byl použit nástroj Lighthouse, který je součástí prohlížeče Google Chrome. Výsledek je zobrazen na obrázku číslo 2.4. Na jednotlivých stránkách aplikace se pohybuje výkonnostní ohodnocení mezi 99 a 100 body. Ostatní metriky mají hodnotu 100. Tyto výsledky ukazují, že aplikace splňuje současné standardy stanovené firmou Google.

3.4 APLIKACE PRO BUDOVY

3.4.1 API specifikace

Jak už bylo zmíněno v sekci 3.1, každá budova by měla mít vlastní server pro sběr a vystavení dat. Jelikož cílem práce bylo vytvořit pouze ukázkové back-end řešení, tak byly zvoleny stejné technologie jako pro front-end aplikaci - tedy Next.js a MongoDB jako databáze. Samotná implementace je tedy pouze ukázková a v reálném nasazení mohou být použity vhodnější technologie, které nemají jako součást frontendovou stránku. Může se jednat třeba o Express nebo dokonce jiný programovací jazyk. Důležité je, aby byla pro každou budovu splněna navržená specifikace, kterou tato ukázková aplikace implementuje.

Každý server pro budovu musí v současné API specifikaci implementovat metody POST a GET. Součástí HTTP požadavků je autorizační hlavička obsahující API klíč, která je pro oba požadavky stejná.

```
{.
    "message": "Service available",
    "measurements": [
        "temperatureInPrague",
        "smartStripCurrent",
        "smartStripVoltage"
    ],
    "aggregationMethods": [
        "$sum",
        "$avg",
        "$min",
        "$max"
    ],
    "lastChange": "2024-03-20T16:08:00Z" }
```

Zdrojový kód 2: Ukázkové tělo odpovědi GET požadavku

Metoda GET slouží k získání informací o konfiguraci. Tento požadavek se používá při vytváření koncového bodu v hlavní aplikaci. Dále je volán před každým POST požadavkem, aby ověřil, zda se konfigurace od posledního požadavku nezměnila, a zda jsou stále k dispozici agregační metody a jednotlivá měření. Tělo odpovědi je ve zdrojovém kódu číslo 2. Nese informaci o tom, zda je služba k dispozici, pole dostupných měření, pole dostupných agregačních metod a datum poslední změny.

Metoda POST vrací agregovaná data na základě těla požadavku. Ukázka je ve zdrojovém kódu číslo 3. Požadavek obsahuje jméno měření, požadovaný rozsah, počet požadovaných bodů měření a metodu pro jejich agregaci.

Odpověď je pak pole objektů, kde každý objekt má hodnotu a časovou značku.

```
Request: {
    "measurementName":"smartStripCurrent",
    "from":"2024-03-17T12:29:24.826Z",
    "to":"2024-03-18T12:29:24.826Z",
    "numberOfItems":2,
    "aggregationOperation":"$avg"
}
Answer:{
    "result": [{
            "value": 0,
            "timestamp": "2024-03-18T12:29:24.826Z"
        },{
            "value": 86,
            "timestamp": "2024-03-19T00:29:24.826Z"}]}
```

Zdrojový kód 3: Ukázkové tělo POST požadavku a odpovědi

Server může implementovat libovolné metody agregace, ale vždy by měl vrátit požadovaný počet hodnot. Pokud data nejsou k dispozici například z důvodu chybějících měření, měla by být vrácená hodnota null.

3.4.2 Agregační metody

Agregace dat může být prováděna na třech místech, a to v hlavní aplikaci, v aplikaci pro jednotlivé budovy nebo přímo v databázi. Jak bylo zmíněno v úvodu kapitoly 3.1, tak je jako databáze použitá MongoDB. Tato databáze podporuje agregaci dat pomocí Agregačního potrubí (Aggregation Pipeline). Toto potrubí umožňuje různé operace sdružování dat, včetně průměrování, sumace, nalezení minimálních a maximálních hodnot v definovaném rozsahu.

Vzhledem k tomu, že ukázková aplikace běží na platformě Vercel a databáze je hostovaná v cloudu pomocí Mongo Atlas, bylo ideální provádět agregaci přímo v databázi. To je z důvodu, že databáze a server budovy nejsou umístěny na stejném místě a databáze vrátí pouze požadovaná data. Pro ukázkovou aplikaci bylo toto řešení nejvhodnější, protože základní členství bylo dostatečně výkonné, aby zvládlo zpracovávat požadavky na agregaci bez problémů. Nicméně, toto řešení nemusí být vždy optimální. Každý případ je individuální a například větší požadavky na výpočetní výkon databáze mohou být ve výsledku dražší než přesunutí více dat a provedení agregace na serveru budovy.

Samotná agregace se skládá ze tří fází. V první fázi takzvané „match stage“ neboli fázi filtrace se vybere požadovaný rozsah dat na základě data od a do. V druhé fázi „group stage“ neboli fázi agregace se vybraná da-

ta sdruží do jednotlivých skupin pomocí vybrané agregační metody a každá skupina obsahuje hodnotu, časovou známku označující začátek intervalu a také identifikátor kolikátý interval je to od počátečního data. V poslední fázi se výsledky setřídí. Na serveru budovy jsou doplněna data o časových intervalech, které byly vynechány kvůli absenci měření. V těchto případech jsou hodnoty nastaveny na null, což jednoznačně označuje, že v daném časovém intervalu nebyly zaznamenány žádné údaje.

3.4.3 Data

Ukázková aplikace obsahuje generování falešných dat i práci s reálnými daty. Do databáze jsou pravidelně ukládána data z chytré zásuvky a údaje o teplotě z Prahy.

Jako generátor náhodných dat byla napsána funkce jejímž vstupem jsou parametry specifikující typ měření, časový rozsah měření v podobě data a času začátku a konce, a počet hodnot, které mají být generovány. Typy měření jsou stejné jako v případě reálných dat, tedy proud, napětí a teplota v Praze. Každé z těchto měření má jiný rozsah hodnot. Pro zajištění opakovatelnosti generovaných dat vzhledem k časovému rozsahu je použit součet UNIXových časů od/do jako vstupní hodnota pro generátor náhodných čísel. Tím je zajištěno, že generovaná data budou reprodukovatelná pro stejný časový rozsah.

Byla použita chytrá zásuvka od společnosti Gosund, která umožňuje měřit napětí a proud v zásuvce. Pro přístup k aktuálním datům byla využita cloudová platforma společnosti Tuya, neboť společnost Gosund nenabízí přímý přístup k datům. Nejprve bylo nutné propojit zásuvku s Tuya aplikací namísto nativní aplikace firmy Gosund. Poté byl vytvořen vývojářský účet na Tuya cloud a propojen s klasickým Tuya účtem. Po úspěšném připojení byl autorizován přístup k zásuvce a byl vygenerován API klíč. Pomocí dokumentace [17] byla vytvořena singleton třída TuyaCloudApiHandler pro komunikaci s API. Tato třída obsahuje metody pro získání a obnovu autentizačního tokenu, neboť API využívá autentizační protokol OAuth 2.0. Dále tato třída obsahuje metody pro získání samotných dat z chytré zásuvky.

Pro sběr dat o počasí byla zvolena integrace Open Weather API, která poskytuje širokou škálu meteorologických údajů, včetně aktuálních teplot, srážek a dalších parametrů. V rámci nejnižší úrovně tarifu Open Weather API je dostupných milion dotazů měsíčně zdarma, což poskytuje dostatečnou kapacitu pro zaznamenávání teploty v určené lokalitě, v tomto případě v Praze. Pro pravidelný sběr dat a jejich ukládání do databáze byl implementován CRON job. Jedná se o pravidelně spouštěný proces, který vykonává určité operace v definovaných časových intervalech.

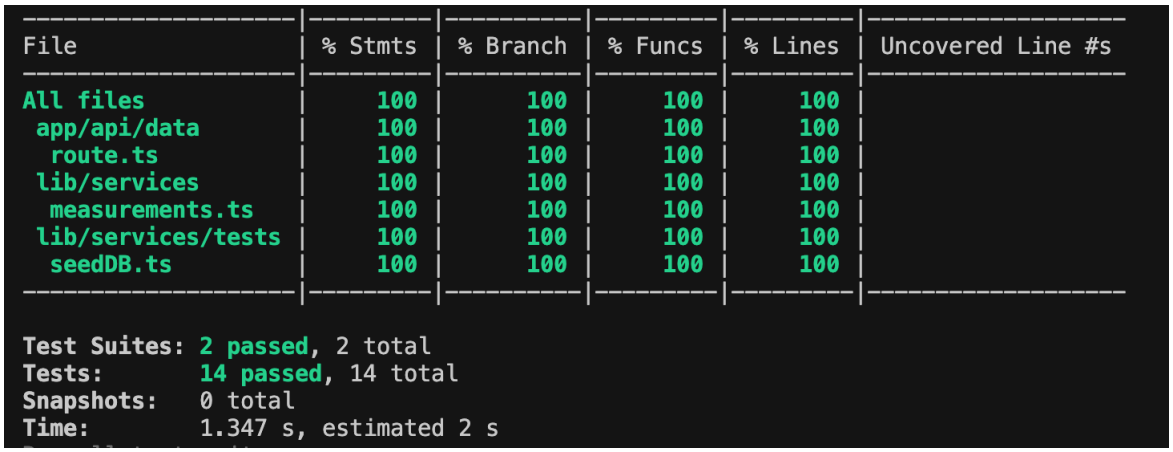
Aplikace je nasazena na platformě Vercel, kde je k dispozici pouze omezený počet CRON jobů a to dva s intervalem spouštění jednou za 24 hodin. Pro pokrytí požadavků na pravidelný sběr dat byla proto zvolena externí služba cron-job.org. Tato služba umožňuje vytváření a správu CRON jobů s vyšší

flexibilitou a počtem spouštění. Konkrétně je možné vytvořit CRON job s periodou opakování jedné minuty. Služba rovněž umožňuje monitorovat posledních 25 odeslaných požadavků, sledovat metriky jako je doba trvání požadavků, jitter a úspěšnost požadavků. Pokud požadavek selže 15krát v řadě, je CRON job automaticky zastaven.

V rámci implementace byl vytvořen CRON job, který periodicky každých 5 minut odesílá POST požadavek na server budovy. Tento požadavek obsahuje přístupový klíč, který byl přidělen pouze CRON jobu a slouží k autentizaci. Neumožňuje tedy přístup k jiným zdrojům serveru.

3.4.4 Testování

Byly napsány testy, které pokrývají práci s uloženými měřeními v databázi a funkčnost API. Jako testovací framework byl použit Jest. Je například testována korektní funkčnost agregačního potrubí. Pokrytí testy je zobrazeno na obrázku číslo 3.14.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
app/api/data	100	100	100	100	
route.ts	100	100	100	100	
lib/services	100	100	100	100	
measurements.ts	100	100	100	100	
lib/services/tests	100	100	100	100	
seedDB.ts	100	100	100	100	

```
Test Suites: 2 passed, 2 total
Tests:       14 passed, 14 total
Snapshots:   0 total
Time:        1.347 s, estimated 2 s
```

Obrázek 3.14: Pokrytí testy aplikace pro budovu

Pro testování API byla spolu s frameworkem Jest použita knihovna „next--test-api-route-handler“, která slouží k usnadnění jednotkového testování API v aplikacích postavených na frameworku Next.js. Byly otestovány požadavky POST a GET, které jsou nutné pro korektní fungování s hlavní aplikací. Tyto testy jsou součástí CI/CD pipeline, která je popsaná v následující kapitole.

3.5 NASAZENÍ

Nasazení proběhlo v následujících krocích. Nejprve byl přidán repozitář z Githubu do Vercel účtu, což znamená, že byla udělena práva k přístupu. Dále bylo potřeba nakonfigurovat projekt: vybrat jméno a přidat proměnné

jako jsou například přístupové údaje do MongoDB. Poté už se mohlo zahájit samotné sestavování projektu.

Propojení Githubu a Vercelu vytvoří základní CI/CD pipeline. CI/CD neboli continuous integration a continuous delivery je série kroků potřebná pro dodání nové verze softwaru. V tomto případě to znamená že Vercel sleduje změny v projektu na Githubu a pokud se objeví nový commit, tak se ho Vercel automaticky pokusí sestavit. To reprezentuje integrační část. Pokud se jedná o commit v hlavní větvi, tak ho v případě úspěšného sestavení rovnou nasadí na produkci - to reprezentuje delivery část.

Vercel dělá předběžné nasazení (preveiew deploy) pro každý commit, a proto v případě pull requestu je možné ověřit, zda je výsledné sestavení úspěšné a až na jeho základě se rozhodovat, zda slučovat větve či nikoliv.

Integrační část byla rozšířena o automatické testování pomocí Github actions. Byl vytvořen soubor, který definuje, že pokud se vytvoří pull request na main větev, tak se vytvoří testovací prostředí a spustí se testy. Zároveň byla nastavena pravidla pro ochranu hlavní větve. Do main větve se nedá pushovat na přímo a musí se nejprve vytvořit pull request a zároveň pro sloučení větví musí mít PR úspěšné testy a sestavení na Vercelu.

Samotná aplikace je pak dostupná na adrese „`https://bms-kesler.vercel.app/`“

4 SHRNUTÍ VÝSLEDKŮ A DISKUZE

Na základě provedené rešerše byly zvoleny vhodné nástroje pro tvorbu aplikace. Jediným problémem vzhledem k vybraným nástrojům byla absence volně dostupné komponenty pro výběr časového intervalu v MUI. Tento problém byl ale vyřešen s využitím další knihovny, která tuto funkcionalitu implementovala a vizuálně zapadala i do celkového konceptu aplikace.

Tomuto problému by se dalo předejít použitím jiné komponentní knihovny, například Ant Designu, který potřebnou komponentu obsahuje zdarma. Použití Ant Designu by bylo ale za cenu horší dokumentace, která například nepopisuje, jak správně pracovat s přístupností vzhledem ke komponentám.

Obě vytvořené aplikace, tedy aplikace pro vytváření vizualizací a aplikace sloužící jako ukázka serveru budovy fungují korektně a byly otestovány pomocí automatických, ale i ručních testů. Ruční testování zabralo nemalé množství času a vzhledem k dalšímu vývoji by bylo vhodné co největší část automatizovat.

Rozdělení řešení na část pro budovy a část pro vizualizace je z pohledu dalšího vývoje vhodná. Existuje nepřeberné množství senzorů a chytrých zařízení, které je možné v budově provozovat. Tyto chytré prvky se liší nejen dostupnými daty ale i komunikačními protokoly. Není tedy možně vytvořit elegantní řešení, ke kterému by bylo možné jednoduše připojit jakékoliv zařízení a pracovat z jeho daty.

Aplikace serveru budovy slouží tedy jako ukázková a implementuje rozhraní pro komunikaci s vizualizační aplikací. Pokud člověk integrující chytré prvky do budovy splní požadavky stanovené v kapitole 3.4.1, je možné využívat vizualizační aplikaci a vytvářet vizualizace na základě dat z koncových bodů. Další vývoj se tedy může soustředit čistě na zlepšování vizualizační části.

V případě změn ve stanovených požadavcích pro komunikaci by bylo vhodné zavést verzování tohoto API. Verze může být součástí odpovědi na GET požadavek, který zjišťuje konfiguraci serveru budovy. Řešením současné absence verze API je označení nynější verze jako jedna a další verze už musí mít tuto informaci jako povinnou.

Nejdůležitější část vytvořené aplikace jsou samotné interaktivní vizualizace. Uživatel má k dispozici čtyři rozdílné možnosti, jak jednotlivá data zobrazit a může je sdružovat v libovolných dashboardech. V rámci dashboardů

může interaktivně upravovat velikost a pozici jednotlivých grafů a tabulek. Lze také v upravovat zobrazený časový interval. Pro zobrazení dat o počasí a z chytré zásuvky byly tyto možnosti vizualizace dostačují. Součástí zadání této práce nebyla specifikace, jaká všechna zobrazení mají být podporována. Další vývoj by se měl ale zaměřit na podporu co nejširšího množství různých vizualizací, které vzniknou jako potřeby koncových uživatelů aplikace.

Bezplatné úrovně poskytované službami Vercel a Mongo Atlas jsou dostatečné pro současný provoz aplikace. Díky využití těchto cloudových služeb, lze řešení jednoduše škálovat. Jak bylo zmíněno v kapitole 3.5, je možné tuto aplikaci provozovat i v Docker kontejneru na jiné platformě. V budoucnu je tedy důležité si spočítat, zda s rostoucím počtem uživatelů není vhodnější z hlediska financí provoz na jiné platformě.

5 ZÁVĚR

V rámci řešení vznikly dvě aplikace. První menší aplikace slouží jako ukázková implementace serveru budovy. Tento server poskytuje testovací ale i reálná data. Jsou sbírána data o teplotě v Praze a údaje o proudu a napětí z chytré zásuvky.

Sbíraná data jsou využita v rámci ukázky v druhé aplikaci, která slouží jako vizualizační. V této aplikaci má uživatel možnost si vytvořit účet a po přihlášení přidávat koncové body.

Koncové body obsahují informace pro připojení k serverům budov. Uživatel tak může přidávat libovolné množství budov. Na základě dat z koncových bodů budov je možné tvořit vizualizace. Stavebním kamenem jednotlivých vizualizací jsou takzvané dashboardy.

Dashboard je místo, do kterého se dají přidávat grafy a tabulky. V rámci interaktivní části dashboardu je možné jednotlivé prvky posouvat a měnit jim velikost. V dashboardu je možné nastavovat i časové rozmezí, která mají jednotlivé tabulky a grafy zobrazit.

Uživatel má na výběr ze čtyřech možností vizualizace dat: sloupcový graf, spojnicový graf, tabulka a interaktivní kombinace tabulky se spojnicovým grafem. Každý z těchto prvků obsahuje titulek a počet bodů, které má zobrazit. Volitelně je možné přidat popisy jednotlivých os. Pro zobrazení dat o počasí a z chytré zásuvky byly tyto možnosti vizualizace dostačují.

V rámci řešení vznikl popis API, který musí server budovy splňovat, aby s ním mohla vizualizační aplikace korektně komunikovat.

V této diplomové práci bylo vytvořeno jednoduché ale přesto komplexní řešení pro správu budovy, které může usnadnit sledování důležitých metrik jak na straně obyvatele, tak na straně správce budovy. Aplikace byla implementována a otestována.

Další vývoj by se měl zaměřit na přidání většího množství vizualizací na základě zpětné vazby od uživatelů. Pro zlepšení využití aplikace i pro nezaškolené uživatele by měla být vytvořena funkcionalita pro sdílení dashboardů. Následně by bylo vhodné vytvořit jazykové mutace aplikace, aby byla aplikace dostupná co nejširšímu publiku. Vhodným doplňkem by pak byla i prezentační webová stránka, která popisuje projekt, jeho fungování a budoucí směřování.

ZDROJE

[1] HIMSCHOOT, Peter. Microsoft Blazor. Online. Berkeley, CA: Apress, 2020. ISBN 978-1-4842-5927-6. Dostupné z: https://doi.org/10.1007/978-1-4842-5928-3. [cit. 2023-10-01].

[2] HAAS, Andreas; ROSSBERG, Andreas; SCHUFF, Derek L.; TITZER, Ben L.; HOLMAN, Michael et al. Bringing the web up to speed with WebAssembly. Online. In: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY, USA: ACM, 2017, s. 185-200. ISBN 9781450349888. Dostupné z: https://doi.org/10.1145/3062341.3062363. [cit. 2023-10-01].

[3] LITVINAVICIUS, Taurius. Exploring Blazor. Online. Berkeley, CA: Apress, 2019. ISBN 978-1-4842-5445-5. Dostupné z: https://doi.org/10.1007/978--1-4842-5446-2. [cit. 2023-10-01].

[4] MARTIN, Robert C. Clean code: a handbook of agile software craftmanship. Robert C. Martin series. Upper Saddle River, NJ: Prentice Hall, c2009. ISBN 0-13-235088-2.

[5] SCHWARZMÜLLER, Maximilian. React Key Concepts. 1. Packt Publishing, 2022. ISBN 978-1-80323-450-2.

[6] Building Management System Application Sheet. Online. In: Optergy. C2023. Dostupné z: https://optergy.com/wp-content/uploads/2024/03/Optergy-AS-Building-Management-System-BMS-Rev2.pdf. [cit. 2024-05-06].

[7] BEIT S.R.O. Bytové Konto. Online. C2015-2024. Dostupné z: https://www.bytovekonto.cz/. [cit. 2024-05-06].

[8] WebAssembly. Online. 2017. Dostupné z: https://webassembly.org/. [cit. 2024-05-06].

[9] Octoverse: The state of open source and rise of AI in 2023. Online. In: DAIGLE, Kyle. Github blog. C2024. Dostupné z: https://github.blog/2023--11-08-the-state-of-open-source-and-ai. [cit. 2024-05-06].

[10] KROTOFF, Tanguy. Frontend Frameworks Popularity. Online. In: Github Gist. C2024. Dostupné z: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190. [cit. 2024-05-06].

[11] MOZILLA. Asm.js. Online. 21 March 2013n. l. Dostupné z: http://asmjs.org. [cit. 2024-05-06].

[12] Web Content Accessibility Guidelines (WCAG) 2.1. Online. World Wide Web Consortium. Dostupné z: https://www.w3.org/TR/WCAG21/. [cit. 2024-05-06].

[13] GOOGLE. Material Design. Online. Dostupné z: https://m3.material.io/. [cit. 2024-05-06].

[14] DEVELOPER EXPRESS INC. DevExpress. Online. C1998-2024. Dostupné z: https://www.devexpress.com/. [cit. 2024-05-06].

[15] ANT GROUP. Ant Design. Online. Dostupné z: https://ant.design/. [cit. 2024-05-06].

[16] VERCEL INC. Vercel. Online. C2024. Dostupné z: https://vercel.com/. [cit. 2024-05-06].

[17] Tuya Developer Platform. Online. TUYA INC. Tuya. C2024. Dostupné z: https://developer.tuya.com/en/docs/iot/. [cit. 2024-05-06].

[18] Can I use WebAssembly. Online. DEVERIA, Alexis a SCHOORS, Lennart. Can I use. Dostupné z: https://caniuse.com/wasm. [cit. 2024-05-06].

[19] StatCounter. Online. C1999-2024. Dostupné z: https://gs.statcounter.com/. [cit. 2024-05-06].

[20] HARDT, Dick. et al OAuth 2.0. Online. 2012. Dostupné z: https://datatracker.ietf.org/doc/html/rfc6749. [cit. 2024-05-06].

[21] UseForm documentation. Online. React Hook Form. 2019. Dostupné z: https://react-hook-form.com/docs/useform#mode. [cit. 2024-05-06].

[22] JETBRAINS S.R.O. JavaScript Programming - The State of Developer Ecosystem in 2023. Online. JetBrains. C2000-2024. Dostupné z: https://www.jetbrains.com/lp/devecosystem-2023/javascript/. [cit. 2024-05-06].

[23] Comparison @angular/core vs react vs vue. Online. POTTER, John. Npmtrends.com. Dostupné z: https://npmtrends.com/@angular/core-vs-react-vs-vue. [cit. 2024-05-06].

[24] VERCEL, INC. Next.js. Online. C2024. Dostupné z: https://nextjs.org/docs. [cit. 2024-05-06].

[25] Material UI. Online. C2024. Dostupné z: https://mui.com/. [cit. 2024--05-06].

[26] TAILWIND LABS. Tailwind CSS. Online. 2017. Dostupné z: https://tailwindcss.com/. [cit. 2024-05-06].

[27] SASS TEAM. Sass: Syntactically Awesome Style Sheets. Online. C 2006–2024. Dostupné z: https://sass-lang.com/. [cit. 2024-05-06].

[28] META PLATFORMS, INC. StyleX. Online. C2024. Dostupné z: https://stylexjs.com/. [cit. 2024-05-06].

[29] Styled components. Online. Dostupné z: https://styled-components.com/. [cit. 2024-05-06].

[30] Cascading Style Sheets. Online. World Wide Web Consortium. Dostupné z: https://www.w3.org/Style/CSS/. [cit. 2024-05-06].

[31] DIGITÁLNÍ A INFORMAČNÍ AGENTURA. Design systém gov.cz. Online. C2023. Dostupné z: https://designsystem.gov.cz/. [cit. 2024-05-06].

[32] SAMOILENKO, Vladimir. MatBlazor. Online. C2018. Dostupné z: https://www.matblazor.com/. [cit. 2024-05-06].

[33] CSS Containment Module Level 3. Online. World Wide Web Consortium. Dostupné z: https://www.w3.org/TR/css-contain-3/. [cit. 2024-05-06].

[34] Nested file conventions and component hierarchy. Online. In: Next.js. C2024. Dostupné z: https://nextjs.org/_next/image?url=%2Fdocs%2Fdark%2Fnested-file-conventions-component-hierarchy.png&w=3840&q=75. [cit. 2024-05-06].

[35] Bytové konto Mockup. Online. In: BEIT S.R.O. Bytové Konto. C2015--2024. Dostupné z: https://www.bytovekonto.cz/wp-content/uploads/2020/07/mockups.png. [cit. 2024-05-06].

[36] EXPECT BEST. Low Angle View of Office Building Against Sky. Online. In: Pexels. C2024. Dostupné z: https://www.pexels.com/photo/low-angle--view-of-office-building-against-sky-323705/. [cit. 2024-05-09].

[37] REED, Samuel. React-grid-layout. Online. NPM. 2016. Dostupné z: https://www.npmjs.com/package/react-grid-layout/v/1.4.1. [cit. 2024-05--10].

[38] MAJ, Wojciech. React datetimerange picker. Online. [2018], [2024]. Dostupné z: https://projects.wojtekmaj.pl/react-datetimerange-picker/. [cit. 2024-05-10].

[39] Express Nodejs Introduction. Online. Mozilla. C1998-2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction. [cit. 2024-05-13].

A LICENCE A VERZE POUŽITÝCH KNIHOVEN

Tabulka A.1: Použité knihovny v aplikaci pro budovy

Knihovna	Licence	Verze
ajv	MIT	8.12.0
axios	MIT	1.6.7
crypto-js	MIT	4.2.0
jsonwebtoken	MIT	9.0.2
mongodb	Apache-2.0	6.3.0
mongoose	MIT	8.2.1
next	MIT	14.1.0
pure-rand	MIT	6.1.0
qs	BSD-3-Clause	6.11.2
react	MIT	18.2.0
react-dom	MIT	18.2.0
tuya-cloud-sdk-nodejs-ex	MIT	1.1.2
@types/jest	MIT	29.5.12
@types/jsonwebtoken	MIT	9.0.5
@types/node	MIT	20.11.6
@types/qs	MIT	6.9.11
@types/react	MIT	18.2.48
@types/react-dom	MIT	18.2.18
@typescript-eslint/parser	BSD-2-Clause	5.62.0
eslint	MIT	8.57.0
eslint-config-next	MIT	14.1.0
eslint-config-prettier	MIT	9.1.0
eslint-plugin-simple-import-sort	MIT	12.0.0
eslint-plugin-unused-imports	MIT	3.1.0
jest	MIT	29.7.0
mongodb-memory-server	MIT	9.1.7
next-test-api-route-handler	MIT	4.0.5
prettier	MIT	3.2.5
prettier-eslint	MIT	16.3.0
typescript	Apache-2.0	4.9.5

Tabulka A.2: Použité knihovny ve vizualizační aplikaci

Knihovna	Licence	Verze
@emotion/react	MIT	11.11.4
@emotion/styled	MIT	11.11.5
@fontsource/roboto	Apache-2.0	5.0.13
@mui/icons-material	MIT	5.15.17
@mui/material	MIT	5.15.17
@mui/x-charts	MIT	7.4.0
@mui/x-data-grid	MIT	7.4.0
@wojtekmaj/react-datetimerange-picker	MIT	5.5.0
aws4	MIT	1.12.0
bcrypt	MIT	5.1.1
bcryptjs	MIT	2.4.3
crypto-js	MIT	4.2.0
eslint-config-prettier	MIT	9.1.0
eslint-plugin-simple-import-sort	MIT	12.1.0
gridstack	MIT	10.1.2
mongoose	MIT	8.3.4
next	MIT	14.2.3
next-auth	ISC	4.24.7
react	MIT	18.3.1
react-dom	MIT	18.3.1
react-grid-layout	MIT	1.4.4
react-hook-form	MIT	7.51.4
@types/bcryptjs	MIT	2.4.6
@types/crypto-js	MIT	4.2.2
@types/jest	MIT	29.5.12
@types/node	MIT	20.12.11
@types/react	MIT	18.3.2
@types/react-dom	MIT	18.3.0
@types/react-grid-layout	MIT	1.3.5
@typescript-eslint/parser	BSD-2-Clause	5.62.0
autoprefixer	MIT	10.4.19
eslint	MIT	8.57.0
eslint-config-next	MIT	14.0.4
eslint-plugin-unused-imports	MIT	3.2.0
jest	MIT	29.7.0
jest-environment-node	MIT	29.7.0
mongodb-memory-server	MIT	9.2.0
postcss	MIT	8.4.38
prettier	MIT	3.2.5
prettier-eslint	MIT	16.3.0
sass	MIT	1.77.1
tailwindcss	MIT	3.4.3
typescript	Apache-2.0	4.9.5

B ZDROJOVÝ KÓD

Součástí práce je zdrojový kód obou vytvořených aplikací, přiložený v archivu sourceCode.zip.