České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra měření

Obor: Inteligentní budovy

Interaktivní energetický dashboard budovy UCEEB

Interactive energy dashboard of the UCEEB building

DIPLOMOVÁ PRÁCE

Vypracoval: Bc. Jan Krčil
Vedoucí práce: Ing. Vít Janovský
Rok: 2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Krčil** Jméno: **Jan** Osobní číslo: **491891**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Inteligentní budovy**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Interaktivní energetický dashboard budovy UCEEB

Název diplomové práce anglicky:

Interactive energy dashboard of the UCEEB building

Pokyny pro vypracování:

Cílem práce je tvorba interaktivního digitálního dashboardu budovy UCEEB. Dashboard má přehlednou formou prezentovat energetické a jiné ukazatele (kvalita vnitřního prostředí apod.) návštěvníkům budovy z řad laické veřejnosti.
Realizace navazuje na projekt, v rámci kterého vznikly neinteraktivní obrazovky dashboardu pro energetiku a kvalitu vnitřního prostředí. Interaktivní dashboard bude v rámci komentovaných prohlídek promítán na dotykovou tabuli. Ta umožňuje interakci s uživatelem.
Realizujte řešení s důrazem na uživatelskou přívětivost, snadné ovládání a přehlednost zobrazených dat. Umožněte uživateli přepínat mezi obrazovkami, měnit časový interval (datum a hodinu) pro zobrazení dat a otevírat podrobné informace např. o jednotlivých místnostech nebo veličinách.
Aplikace dashboardu:
• je instalován na serveru UCEEB včetně komponent,
• prostřednictvím různých API získává data ze senzorů a provozu budovy,
• pro rychlejší načítání a optimalizaci si ukládá lokálně kopii dat min za poslední rok,
• je připravena pro prezentaci na interaktivní tabuli, webových stránkách apod.
• umožňuje interakci s uživatelem (měnit časový interval dat, zobrazit podrobnosti…),
• je dobře zdokumentována a kód je okomentován, aby bylo možné dále aplikaci rozšiřovat a upravovat.
Proveďte moderované uživatelské testování (min 10 respondentů) s cílem zhodnotit přehlednost a přívětivost dashboardu. Při moderovaném uživatelském testování respondent prochází dashboard podle připraveného scénáře.

Seznam doporučené literatury:

[1] Bach, B. et al., Dashboard Design Patterns, IEEE Transactions on Visualization and Computer Graphics, vol. 29, no. 1, pp. 342-352, Jan. 2023, doi: 10.1109/TVCG.2022.3209448
[2] Human-Computer Interaction. Theory, Methods and Tools: Thematic Area, Proceedings, Part I, edited by Masaaki Kurosu, Springer International Publishing AG, 2021
[3] Minichino, J., Data Analytics in the AWS Cloud: Building a Data Platform for BI and Predictive Analytics on AWS, John Wiley & Sons, Incorporated, 2023. ProQuest Ebook Central

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Vít Janovský katedra informačních a komunikačních technologií v lékařství FBMI

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.02.2024** Termín odevzdání diplomové práce: **24.05.2024**

Platnost zadání diplomové práce:
do konce letního semestru 2024/2025

Ing. Vít Janovský podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne ....

Bc. Jan Krčil

Poděkování

Tímto bych rád poděkoval Ing. Vítu Janovskému za pomoc při vypracování diplomové práce. Také bych chtěl poděkovat svojí rodině a blízkým za podporu po celou dobu mého studia na ČVUT.

Bc. Jan Krčil

Název práce:

Interaktivní energetický dashboard budovy UCEEB

Autor: Bc. Jan Krčil

Studijní program: Inteligentní budovy
Obor: Inteligentní budovy
Druh práce: Diplomová práce

Vedoucí práce: Ing. Vít Janovský
UCEEB

Abstrakt: Tato práce popisuje podrobnou analýzu dostupných technologií, požadavků a tvorbu aplikace dashboardu určeného ke shromažďování a zobrazování dat z různých senzorů instalovaných v budově UCEEB. Studie zkoumá technické aspekty sběru, zpracování a vizualizace dat ze senzorů. Praktická část práce se zaměřuje na konkrétní použité technologie, architekturu aplikace, návrh a implementaci komponent dashboardu. Práci uzavírá testovací fáze, včetně uživatelského průzkumu a následných úprav na základě získané zpětné vazby.
Klíčová slova: UCEEB, dashboard, energy, sensor

Title:

Interactive energy dashboard of the UCEEB building

Author: Bc. Jan Krčil

Abstract: This thesis describes the detailed analysis of available technologies and development of a dashboard application designed to collect and display data from various sensors installed in the UCEEB building. The study examines the technical aspects of collecting, processing and visualizing sensor data. The practical part of the work focuses on the specific technologies used, the architecture of the application, and the design and implementation of the dashboard components. The thesis concludes with a testing phase, including a user survey and subsequent modifications based on the feedback received.
Key words: UCEEB, dashboard, energy, sensor

Obsah

Seznam použitých zkratek

IoT	Internet of things
UCEEB	Univerzitní centrum energeticky efektivních budov
API	Application programming interface
MVC	Model-View-Controller
ORM	Object relational mapping
ACID	atomicity, consistency, isolation and durability
IAQ	Indoor air quality
WSGI	Web server gateway interface
SQL	Structured query language
XML	Extensible Markup Language
JSON	JavaScript Object Notation
DOM	Document object model
HTTP	Hypertext Transfer Protocol
CSS	Cascading Style Sheets
REST	Representational State Transfer

Seznam obrázků

Úvod

V rámci rozvoje IoT[1] a obecně senzoriky v inteligentních budovách se stále častěji setkáváme s možností monitorovat různé aspekty budovy od teploty a vlhkosti až po spotřebu energie. Aby bylo možné z těchto dat získat nějakou informaci užitečnou například pro návštěvníka budovy nebo jejího správce, je třeba je zpracovat a v uživatelsky přívětivém formátu zobrazit.

Cílem práce je navrhnout a implementovat interaktivní dashboard pro budovu UCEEB[2] zobrazující energetické a jiné ukazatele ze senzorů vně a uvnitř budovy. Vzhledem k tomu, že budova sama o sobě slouží jako laboratoř a jsou zde prováděny pokusy, byla vytvořena poptávka po dashboardu, který by byl umístěn do vstupu budovy a zejména návštěvnici ale i zaměstnanci UCEEBu by ho mohli využít. Dashboard by měl být přístupný z webového prohlížeče a měl by být zaměřen spíše na laickou veřejnost. Prioritou je tedy čitelnost a srozumitelnost.

Práce je strukturována následujícím způsobem: Teoretická část obsahuje analýzu a popis procesu návrhu uživatelského prostředí dashboardu. Praktická část popisuje použité technologie, architekturu aplikace, návrh a implementaci komponent dashboardu. Práci uzavírá část o uživatelském testování včetně uživatelského průzkumu a následných úprav na základě získané zpětné vazby. Tím, že tato práce poskytuje praktické řešení problému vizualizace dat ze senzorů v budovách, přispívá k rozvoji rostoucí oblasti technologií inteligentních budov a má dopad na jejich správu.

[1] Internet of things

[2] Univerzitní centrum energeticky efektivních budov

Část I

Teoretická část

Kapitola 1

Analýza

Tato část je zaměřena na analýzu zadání práce, požadavků na dashboard a dále pak na rozbor možných technických řešení vhodných pro tento účel. Byla z většiny vypracována v rámci Projektu 2.

1.1 Definice dashboardu

Typickým příkladem dashboardu je přístrojová deska v autě nebo v letadle, která zobrazuje zásadní informace ohledně rychlosti, tlaku oleje, teploty a dalších. Světla, ukazatele a další informace jsou umístěny tak, aby uživatel byl schopen rychle analyzovat informace a jednat na jejich základě. Úplně stejný princip funguje například i u manažerských dashboardů. V případě dashboardu pro UCEEB se jedná o dashboard informační, dal by se označit za analytický. Dle [1] organizace používají analytické dashboardy ke sledování pokroku a souvisejících trendů.

Dalšími běžně používanými typy jsou dashboardy operační, které se používají pro monitorování obchodních procesů, aktivit a komplexních událostí a také dashboardy strategické, pomocí kterých organizace monitorují postup směrem ke strategickým cílům. Většina dashboardů tohoto typu je interaktivních a často umožňuje nejenom zobrazení ale i práci s informacemi a daty.

1.2 Požadavky

V případě běžných webových aplikací se definují dva typy požadavků, a to funkční a nefunkční.

1.2.1 Funkční požadavky

Funkční požadavky jsou seznamem funkcí systému, které je potřeba implementovat, aby uživatelé mohli systém používat. Také popisují chování systému při specifických vstupech od uživatele. V případě tohoto dashboardu jsou funkčními po-

žadavky nároky na konkretní data, která by měl dashboard zobrazovat a možnost upravovat kontext zobrazení.

- Zobrazení dat vnitřní prostředí (teplota, vlhkost, koncentrace CO_2).
- Zobrazení energetických dat jako příkon budovy, výkon fotovoltaické elektrárny atd.
- Grafické zobrazení těchto dat a jejich průběhů v čase.
- Uživatel je schopen měnit časový rozsah grafů.
- Uživatel může měnit momentálně zobrazený senzor.
- Uživatel může libovolně přepínat mezi obrazovkami dashboardu.

1.2.2 Nefunkční požadavky

Seznam vlastností, dle kterých jde analyzovat způsob chování a kvalitu fungování systému. Definují omezení řešení. Tato aplikace má nároky zejména na uživatelskou přívětivost, čitelnost, přehlednost a schopnost předat informace o budově všem jejím návštěvníkům bez ohledu na odbornost.

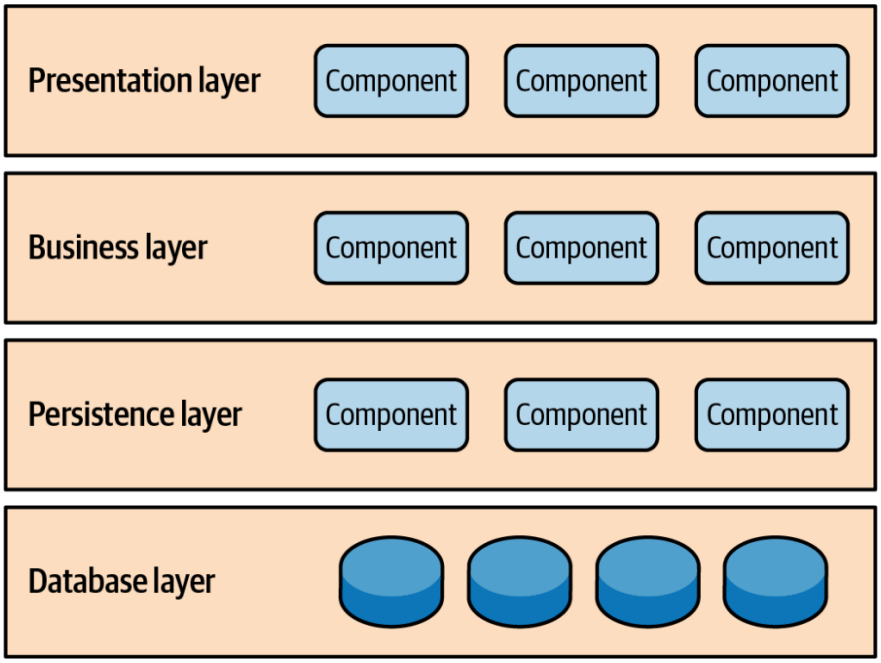
- Uživatelská přívětivost
  - Software je zaměřen především na laickou veřejnost, proto by neměl zobrazovat data ve zbytečně složité formě a neměl by uživatele zahltit informacemi.
- Čitelnost
  - Prezentované informace musí být pochopitelné a snadno čitelné. Vzhledem k umístění dashboardu nebude možnost si jednotlivá data přiblížit nebo jakkoliv zvýraznit.
- Stabilita
  - Pro dashboard tohoto typu je zásadní, aby byly ošetřené výjimky a nedošlo k pádu aplikace.
- Aktualizace dat
  - Data zobrazená na dashboardu se budou periodicky obnovovat. Je důležité zvolit správnou frekvenci obnovování, aby byla data aktuální a zároveň nebylo obnovování rušivé.

1.3 Analýza softwarové architektury

Tato část je věnována analýze architektur softwaru vhodných pro tvorbu dashboardu na základě webové aplikace.

1.3.1 Vrstevnatá architektura

Jedná se o jednu z nejrozšířenějších architektur pro většinu aplikací. Skládá se z komponent, které jsou seřazeny do horizontálních vrstev, z nichž každá plní určitou roli v rámci architektury (například prezentační logika, obchodní logika, persistence, databázová logika apod.) [2].



Obrázek 1.1: Příklad vrstevnaté architektury (převzato z [2])

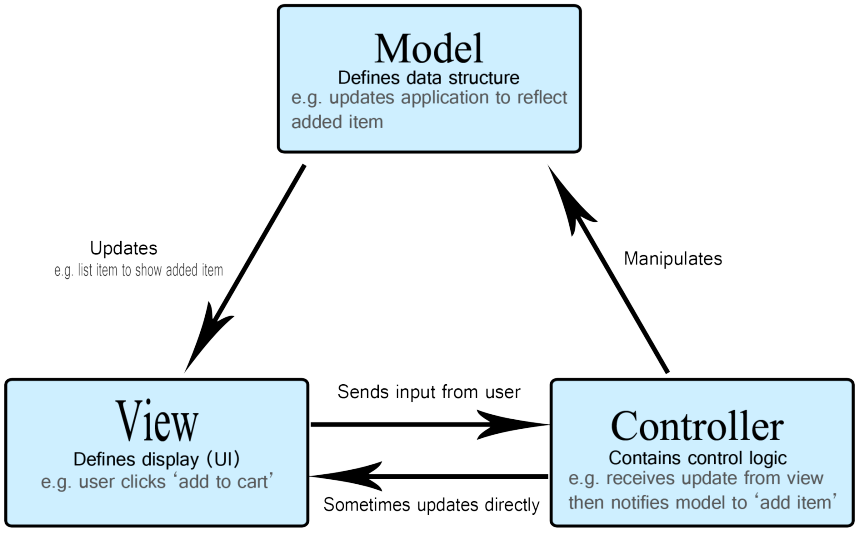
Základní premisou funkce takovéto architektury je dodržování dvou konceptů: izolace vrstev a rozdělení odpovědností. Izolace vrstev znamená, že změny provedené v jedné vrstvě by neměly nijak ovlivňovat vrstvu druhou. Rozdělení odpovědnosti vrstev se projevuje tak, že každá z nich se zabývá pouze jednou „oblastí“ fungování aplikace. Například komponenty v prezentační vrstvě řeší pouze prezentaci dat. Většinou také komponenty v jedné vrstvě mohou komunikovat pouze s vrstvou, která s nimi přímo sousedí. Příklad takového chování by byl při vytváření nového uživatele v aplikaci. V prezentační vrstvě uživatel vyplní své údaje. Business vrstva nebo logická vrstva je zpracuje dle požadavků a persisenční vrstva uloží do databáze. Prezentační vrstva pak zobrazí potvrzení, že byl zákazník vytvořen. Prezentační vrstva nemá žádné „znalosti“ o datech, pouze je zobrazuje. Je zde však i možnost jednu z vrstev přeskočit, pokud by například nebylo třeba data logicky zpracovávat.

Tato architektura je vhodná pro projekty, které nejsou příliš komplexní. Je poměrně jednoduchá a lze se v ní intuitivně orientovat. Problémem této architektury je poměrně špatná škálovatelnost. Vzhledem k vrstvám, které samy o sobě nemohou fungovat, je třeba vždy škálovat celou aplikaci, což je problém u větších projektů. Také není vhodná pro případy kdy je třeba často provádět změny na úrovni domény. Například i tak jednoduchý úkol jako přidání data vypršení platnosti do seznamu filmů v aplikaci na jejich sledování znamená změnu databázového schématu, samotné databáze, změnu v logice aplikace (Limitace pro datum, kdy má vypršet apod.) a následně i změnu prezentační vrstvy, kdy je třeba datum zobrazit.

1.3.2 Model-View-Controller

MVC[1] je architektonický vzor, který se často používá při vývoji uživatelských rozhraní. Je rozdělen do tří hlavních komponent: Model, View a Controller. Model reprezentuje data a pravidla aplikace. Je zodpovědný za ukládání a načítání dat, a také za jejich validaci a zpracování. View zobrazuje data uživateli a je zodpovědný za to, jak jsou data prezentována a zobrazena. Controller řídí interakci mezi modelem a view. Přijímá vstupy od uživatele a provádí odpovídající akce v modelu [3].

MVC architektura má několik výhod. Jednou z nich je oddělení zájmů. MVC odděluje logiku aplikace, uživatelské rozhraní a zpracování dat, což usnadňuje údržbu a testování. Další výhodou je flexibilita. Komponenty MVC lze snadno nahradit nebo upravit bez ovlivnění ostatních částí systému. Nicméně, MVC architektura má také několik nevýhod. Podobně jako u vrstevnaté architektury je jednou z nevýhod výkon. Stejně jako vrstevnatá architektura, i MVC může trpět problémy s výkonem kvůli komunikaci mezi komponentami.



Obrázek 1.2: Příklad návrhového vzoru Model View Controller (převzato z [4])

[1] Model-View-Controller

1.3.3 Mikroservices

Architektura mikroslužeb je ekosystém tvořený jednoúčelovými samostatně nasazenými službami, které jsou obvykle přístupné prostřednictvím API[2] brány (Více k používání a specifikaci API v části 2.3.). Klientské požadavky vycházející buď z uživatelského rozhraní nebo z externího požadavku vyvolávají dobře definované koncové body v API bráně, která pak předává uživatelský požadavek samostatně nasazeným službám. Každá služba přistupuje ke svým vlastním datům nebo požaduje přístup k datům, která nevlastní od jiných služeb [2].

Ačkoli každá služba může být spojena s oddělenou databází, obvykle tomu tak není. Spíše má každá služba vlastní kolekci tabulek obvykle ve formě schématu, které může být umístěno v jedné databázi dostupné všem službám nebo v databázi věnované konkrétnímu doménovému modelu. Klíčovým konceptem je, že pouze služba vlastnící tabulky může přistupovat a aktualizovat tato data. Pokud jiné služby potřebují přístup k těmto datům, musí požádat vlastnickou mikroslužbu o tyto informace, místo aby přistupovaly k tabulkám přímo.

1.4 Technologie

Tato část je věnována analýze možných řešení vhodných pro použití v kontextu dashboardu. Vzhledem k tomu, že dashboard má být kromě obrazovky v lobby budovy UCEEB přístupný i online, bude se jednat o webovou aplikaci a tedy i technologie k tomu vhodné.

1.4.1 Logická část aplikace

Část aplikace která zodpovídá za databázi a logiku. V případě webové aplikace je třeba vybrat framework, což je abstrakce, v níž lze software poskytující obecnou funkčnost selektivně měnit dalším kódem napsaným uživatelem, a tím poskytovat software specifický pro danou aplikaci. Dále zmíněné frameworky mají v sobě často zabudovanou i podporu frontendové části (viz část 1.4.3), která ale není pro vývoj této aplikace tak zásadní, a také se backend poskytovaný těmito frameworky často kombinuje se odděleným frontendem, který s backendem komunikuje pomocí API.

V případě aplikace dashboardu bude backend primárně získávat a zpracovávat data z externích API senzorů na budově a dále předávat uživatelskému rozhraní.

[2]Application programming interface

Flask

Flask je WSGI[3] webový aplikační framework napsaný v Pythonu[4]. Je navržen tak, aby bylo poměrně jednoduché a rychlé začít, ale zároveň aby umožňoval škálovatelnost a podporu složitějších aplikací [5].

Na rozdíl od frameworku Jakarte EE nebo Django nenutí uživatele používat žádné závislosti nebo strukturu projektu. Flask sám o sobě pouze obaluje Jinju (více v kapitole 1.4.3) a Werzeug (WSGI knihovna pro Python). Z toho důvodu je Flask klasifikován jako mikroframework, což znamená, že implementuje pouze základní funkcionalitu, ale pokročilejší funkce jako autentizaci nebo ORM[5] ponechává na knihovnách [6].

Z vlastností mikroframeworku však plynou i jisté nevýhody. Udržování velkých projektů může být poměrně složité a nutnost všechny knihovny manuálně instalovat a spravovat také může být problém. Lze též říct, že komunitní podpora frameworku je lehce horší oproti například frameworku Django.

Django

Framework Django je stejně jako Flask vytvořen v jazyce Python. Byl vytvořen pro rychlý vývoj databázově řízených stránek. Používá je vysokoúrovňovou, open source sadu knihoven ve stylu MVC [7].

Spring

Open source (Program nebo systém s volně přístupným zdrojovým kódem, který tak může každý libovolně upravovat a měnit za účelem vývoje daného nástroje.) framework v jazyce Java[6]. Podobně jako Flask tak i Spring v sobě nemá integrovanou velkou část funkcionalit, lze je však velmi kvalitně nahradit pomocí knihoven. Je třeba zmínit, že Spring má velmi dobrou integraci s technologií Spring Boot, která nabízí většinu funkcionalit, jež jsou obsažené v Djangu.

V porovnání s Djangem a Flaskem má Spring lepší možnosti škálovatelnosti a dalo by se říci, že je vhodnější pro velké enterprise aplikace. Není však tak intuitivní jako alternativní python frameworky a vývoj může být pomalejší. Samotná základní konfigurace aplikace může často být komplikovaná a zdlouhavá.

Co se týká architektury, tak Spring podporuje zejména vrstevnatou architekturu, která rozděluje aplikaci do vrstev podle zaměření a komunikace je sekvenční – vrstvy komunikují pouze se sousedními. Jak taková architektura vypadá, lze vidět na obrázku 1.1.

[3]Web server gateway interface

[4]Vysokoúrovňový programovací jazyk

[5]Object relational mapping

[6]Vysokoúrovňový objektově orientovaný programovací jazyk založený na třídách.

Jakarta EE

Jakarte EE je stejně jako Spring v jazyce Java a také využívá vrstevnatou architekturu (viz obrázek 1.1). Jedna se o ještě robustnější a lépe škálovatelné řešení, než Django nebo Spring. Architektura není tak modulární jako u Springu, spíše monolitická a integrovaná [8]. Díky tomu má jednotný model a standardizované API. To může být kladnou i zápornou vlastností. Pro případ užití této práce by se jednalo o framework spíše nevhodný a zbytečně obsáhlý. Jakarta je vhodnější pro stabilní prostředí, kde je zásadní robustnost a bezpečnost.

Ruby on Rails

Ruby on rails je další framework pro tvorbu webových aplikací, tentokrát založený na objektově orientovaném programovacím jazyce Ruby. Je velmi podobný Djangu v zaměření na rychlost vývoje a prototypování (Proces vývoje funkční repliky produktu nebo systému za účelem získání zpětné vazby od zákazníků, který umožňuje opakované zdokonalování a zlepšování na základě výkonnosti prototypu [9].). Rails je podobně jako Django založen na architektuře MVC, ale upřednostňuje „konvence před konfigurací“, tedy dodržování standardních konvencí usnadňujících vývoj a srozumitelnost a vylučuje složité konfigurační soubory [10]. To má za následek nižší flexibilitu zejména oproti frameworkům jako je Flask nebo Spring. Co se týká škálovatelnosti, tak se řadí přibližně doprostřed pomyslného žebříčku.

1.4.2 Databáze

Výběr databáze významně ovlivňuje fungování celé aplikace. Na nejnižší úrovni lze databáze používané ve webových aplikacích rozdělit na dva typy: relační a nerelační, které se pak dále dělí například na grafové a dokumentové. V případě této práce se jedná o ukládání dat ze senzorů, kdy je třeba uložit samotnou hodnotu a dále informace o ní (Identifikace senzoru, čas kdy bylo měření pořízeno atp.).

Relační databáze

Tyto databáze jsou nejběžnějším typem a data jsou v nich uspořádaná do tabulek, které obsahují informace o jednotlivých entitách a prostřednictvím řádků a sloupců reprezentují předem definované kategorie. K datům lze přistupovat a sestavovat je pomocí dotazů [11]. Příklady relačních databází zahrnují například MySQL[7] nebo PostgreSQL[8]. Jsou ideální pro aplikace, které vyžadují komplexní dotazy a transakce a zároveň poměrně rigitní strukturu dat – jakákoliv informace se musí přizpůsobit tabulce, do které je ukládána, a respektovat její pravidla.

[7] https://www.mysql.com/
[8] https://www.postgresql.org/

Výhodou těchto databází je silná podpora pro transakce, konzistenci dat a integritu dat – garantují vlastnosti ACID[9] (atomicity, consistency, isolation a durability). Atomicity znamená, že transakce je považována za jednotný celek, který celý úspěšně proběhne nebo se v případě výskytu chyby neprovede žádná část transakce. Consistency zajišťuje, že databáze zůstane v konzistentním stavu před a po transakcí. Isolation – paralelně probíhající transakce se navzájem neovlivňují. Durability – jakmile je transakce dokončena, změny v databázi jsou trvalé, i když dojde k systémovému selhání. Nevýhodou je obtížnější horizontální škálovatelnost (rozšiřování na více serverů). Navíc, pevně stanovené schéma může být pro některé typy aplikací omezující.

Nerelační databáze

Nerelační databáze představují typ systémů, které spravují databáze a významně se liší od relačních. Nejdůležitější rozdíl je v tom, že nerelační databáze nepoužívají relace (tabulky) jako svou úložnou strukturu. Také nepoužívají SQL[10] jako svůj dotazovací jazyk, nelze provádět spojovací operace, negarantují vlastnosti ACID a mohou být horizontálně škálovány.

Nerelační databáze mohou být primárně klasifikovány na základě způsobu organizace dat následovně:

1. Úložiště klíč-hodnota: Tento typ databáze umožňuje vývojářům aplikací ukládat data bez schématu. Tato data se skládají z klíče, který je reprezentován řetězcem, a skutečných dat, která jsou hodnotou v páru klíč-hodnota. Data mohou být jakéhokoli typu podporovaného programovacím jazykem, tedy například celé číslo, pole, nebo objekt. Tím se uvolňuje požadavek na formátovaná data pro úložiště, čímž se eliminuje potřeba pevného datového modelu [11]. Tento typ nerelační databáze by teoreticky mohl být použitelný i pro ukládání dat ze senzorů, ne však ideální. K senzorům bude třeba přistupovat přes různé klíče – například datum pořízení, identifikační číslo, typ měření atp. Vzhledem k tomu by tato databáze byla spíše nevhodná.

2. Dokumentové úložiště: Tento typ databáze, také běžně známý jako „dokumentově orientovaná databáze“, je v podstatě počítačový program používaný pro ukládání, načítání, aktualizaci dat uložených v databázi. Základní úložnou strukturou používanou v těchto databázích je dokument. Každé dokumentové úložiště se liší ve své implementaci dat; nicméně každé z nich předpokládá, že data jsou uzavřena a kódována v nějakém standardním formátu, který může být XML[11], JSON[12], PDF nebo Microsoft Office [11].

[9] atomicity, consistency, isolation and durability
[10] Structured query language
[11] Extensible Markup Language
[12] JavaScript Object Notation

3. Grafové databáze jsou databáze bez schématu, které používají grafové datové struktury spolu s uzly, hranami a určitými vlastnostmi k reprezentaci dat. Uzly mohou reprezentovat entity jako jsou lidé, podniky nebo jakékoli jiné položky podobné tomu, co reprezentují objekty v jakémkoli programovacím jazyce. Vlastnosti označují jakékoliv důležité informace související s uzly. Hrany spojují uzel s jiným uzlem nebo uzel s nějakou vlastností. Studiem propojení mezi uzly, vlastnostmi a hranami lze získat nějaký smysluplný vzorec nebo chování [11].

Na základě těchto informací je zřejmé, že tento typ databáze není vhodný na ukládání dat ze senzorů, jelikož se jedná o data s velmi rigidní strukturou, která jsou dlouhodobě stejná. Hlavní výhoda nerelační databází (eliminace pevného datového modelu) tak není relevantní.

1.4.3 Frontend

Uživatelské rozhraní aplikace je v případě analytického/informačního dashboardu zásadní stejně jako vybraná technologie. V základě jsou v současnosti při tvorbě rozhraní webové aplikace dvě možnosti. Server-side rendering, kde se jedná o monolitickou aplikaci běžící na jednom zařízení. Toto řešení často využívá šablon, které jsou integrované v backendových frameworcích jako Flask, Spring nebo Django. Druhou možností je oddělený frontend s využitím jedné z knihoven jako například React, které využívají client-side rendering.

Šablonovací systémy

Příkladem „templating engine“ je Jinja2 použitá ve frameworku Flask nebo Django. Jak již bylo zmíněno, šablony jsou založené na tzv. server-side renderingu, kdy webový prohlížeč odešle požadavek na informace ze serveru, načte data specifická pro uživatele, která vyplní a odešle klientovi plně vykreslenou stránku HTML. Pokaždé, když uživatel navštíví novou stránku na webu, server celý proces zopakuje [12]. Server-side vykreslování je většinou rychlejší než client-side a vyhledávačům se stránka lépe indexuje.

Samotná Jinja2 umožňuje kromě jiného především používání syntax podobnou jazyku Python v rámci stránky a díky tomu podporuje dynamické zobrazovaní variabilních dat – stránka není statická.

React nebo jiná knihovna tohoto typu

React a jemu podobné knihovny využívají client-side rendering, kdy je javascriptový kód vykreslován/prováděn až v prohlížeči a ne na serveru, a to snižuje nároky na server. Díky tomu také react podporuje řadu zajímavých funkcí, zejména „Virutal

DOM“, tedy virtuální reprezentaci DOM[13]. V paměti udržuje odlehčenou kopii skutečného DOM, což mu umožňuje při změnách efektivně aktualizovat a vykreslovat pouze potřebné součásti namísto opětovného vykreslování celé stránky [13]. Také je třeba zmínit, že react je založený na komponentách, tedy rozdělení uživatelského rozhraní na opakovaně použitelné komponenty. Každá komponenta může spravovat svůj vlastní stav a vlastnosti, což usnadňuje vytváření složitých uživatelských rozhraní a údržbu kódu.

1.4.4 Specifické zobrazovací knihovny

Bez ohledu na výběr základní technologie uživatelského rozhraní je třeba vybrat vhodnou knihovnu pro zobrazení dat ve správném kontextu. Níže se nachází analýza uvažovaných běžně používaných knihoven a jejich vlastností.

Fusion charts

Knihovna grafů, která nabízí více než 100 interaktivních grafů a více než 2000 map založených na datech. Grafy jsou poměrně snadno použitelné, vysoce přizpůsobitelné a graficky kvalitně zpracované. Knihovna také disponuje rozsáhlou dokumentací [14]. Problémem fusion charts je pouze částečný přístup bez předplatného a nevzhledné vodotisky.

Chart.js

Odlehčená a snadno použitelná knihovna pro tvorbu grafů. Nabízí animace a přechody dat, podporu data a času, logaritmické funkce a možnost kombinovat různé typy datových grafů. Knihovna má kvalitní anglickou dokumentaci [15]. Velkým problémem této knihovnu je, že grafy vykresluje v html tagu canvas, se kterým se nepracuje příliš dobře, má problémy s responsním designem a výběr grafů je omezený.

Plotly

Výkonný nástroj, který podporuje velké množství grafů, rozhodně větší než například Chart.js, a je plně open-source. Nabízí funkce, jako jsou interaktivní grafy (hover, zoom, pan) a online editor grafů. Také je dostupný ve více jazycích na rozdíl od předchozích knihoven, které jsou založené pouze na javascriptu. Plotly disponuje extenzivní dokumentací popisující všechny možné typy grafů a práci s nimi [16].

[13]Document object model

Kapitola 2

Postup návrhu dashboardu

2.1 Identifikace uživatelů

Jak již bylo zmíněno v předchozí kapitole, primárními uživateli dashboardu bude laická veřejnost, která se bude v budově UCEEB nacházet například za účelem prohlídky. Z tohoto důvodu by měl dashboard zobrazovat informace v pochopitelné jednoduché formě a nepřehlcovat jimi uživatele. Také by ale měl poskytnout zajímavé informace i běžným uživatelům budovy UCEEB.

2.2 Identifikace potřebných informací

Budova UCEEB má na sobě velké množství senzorů, které vevnitř budovy sledují vše od teploty, přes vlhkost a koncentraci CO_2, po rosný bod. Vně se pak měří meteorologická data nebo výkon fotovoltaické elektrárny [17]. Zatímco teplota, relativní vlhkost a koncentrace CO_2 jsou informace, pod nimiž si velká část populace dokáže něco představit, například VOC index (bezrozměrné číslo určující kvalitu vnitřního prostředí [18]) už obecně tak známý není. Z toho důvodu půjde v rámci vnitřního prostředí o zobrazení teploty, vlhkosti a koncentrace CO_2, a to jak současných hodnot tak průběhů. V energetické části dashboardu pak budou zobrazována data z baterie, fotovoltaiky, kogenerační jednotky a hlavního elektroměru budovy.

2.3 Analýza a přístup k datovým API

Data zmíněná v předchozí kapitole bude aplikace získávat z různých HTTP[1] a API napojených na zařízení v a na budově UCEEB. API je protokol, který popisuje, jakým způsobem a kterými metodami lze získat zdroje. HTTP API je typ rozhraní, které využívá zmíněný HTTP protokol. S jeho pomocí je definováno množství tzv.

[1] Hypertext Transfer Protocol

„endpointů“, tedy konkrétních metod přístupu k rozhraní, které aplikace vystavuje za účelem předání dat.

2.3.1 Indoor air quality data

Prvním příkladem rozhraní, které bude použito v práci, je IAQ[2] UCEEB. Toto rozhraní umožňuje přístup k datům ze senzorů vnitřního prostředí budovy, jako je teplota nebo vlhkost v místnostech [19]. Toto rozhraní používá proměnné v URL, jak je vidět na následujícím příkladu, kde se nachází dotaz na konkrétní senzor.

Listing 2.1: Příklad URL dotazu na rozhraní

```
/api/v1/applications/:application_id/devices/
:device_id/sensors/:sensor_id
```

Také je třeba zmínit, že API nepodporuje tzv. „hromadné requesty“ – nelze si vyžádat data z několika senzorů najednou. Toto má za následek, že získávaní většího množství dat může být pomalejší, než je běžné.

2.3.2 Mervis Scada

Mervis Scada poskytuje UCEEBu informace a kontrolu nad systémem vytápění a obecně správy energií. Mervis poskytuje ke své již existující aplikaci webové API. Cílem je z tohoto systému získat data o příkonu budovy a výkonu kogenerační jednotky. API má přehlednou dokumentaci, kdy přístup probíhá nejdříve získáním tokenu, který je pak využíván u dalších dotazů [20]. Parametry dotazu včetně tokenu a požadovaných senzorů jsou předávány v těle dotazu, které lze vidět v listingu 2.2.

Konkrétně tento dotaz přes parametr „dpIds“ získá data ze šesti zdrojů najednou a díky tomu je API velmi rychlé. Další dobrou vlastností je možnost exportování tagů použitých v „dpIds“ a jeho korespondujících senzorů.

Zajímavostí tohoto API je, že podporuje segmentaci, díky které jsou dotazy efektivnější a při práci s větším množstvím dat (například při získání dat z měření za rok). Díky rozdělení načítaných dat na menší části je rozhraní schopné pracovat s postupným zatížením, a zabraňuje se tak vyčerpání paměti a přetížení sítě.

[2]Indoor air quality

Listing 2.2: Definice URL dotazu na rozhraní Mervis

```
url = "https://scada.mervis.info/api/get/values?format=xml"
headers = {'Content-Type': 'application/json'}
body = {
    "cred": {
        "t": api_token
    },
    "propNamesToSerialize": ["Output"],
    "offset": 0,
    "count": 1000,
    "serverState": None,
    "dps": [{
        "projId": "2a7f1615-42f2-44ef-a643-f1438ed44e2a",
        "dpIds": ["be66d396-200b-d0a7-68db-22bdd6adfd6b",
                  "f9e2ae39-e5ed-0c7e-a52f-1547fa748e82",
                  "7119c451-61e6-2250-ff35-beed827bab98",
                  "00078c8e-e8e6-12c7-b392-c91794973382",
                  "553d3193-b406-58eb-f580-f25a45506f34",
                  "d71cabd4-cb8f-9352-aa90-d0982140cf21"
                  ]
    }
    ]
}
```

2.3.3 Solar Edge

Solar Edge je další platforma pro správu energií. V UCEEBu se využívá pro správu fotovoltaické elektrárny na střeše o výkonu 75.864 kWp. Z této elektrárny je potřeba získat data o aktuálním výkonu a průběhu vyrobené energie v čase. K tomuto účelu je zde opět rozhraní, které řešení Solar Edge nabízí. Řešení je velmi funkční, níže je vidět příklad JSON odpovědi při dotazu na celkový přehled elektrárny, který bude použit i v aplikaci [21].

Toto API splňuje požadavky na REST[3]ful rozhraní, což znamená, že dotazy jsou strukturované podle určitého klíče. Používá předvídatelné URL cesty, má v sobě zabudované funkce HTTP pro předávaní parametrů prostřednictvím rozhraní, odpovídá standardními HTTP kódy (např. 200 = success) a umožňuje posílání odpovědi jak ve formátu JSON tak XML.

JSON a XML jsou reprezentace dat používané při jejich výměně mezi aplikacemi [22]. JSON je otevřený formát pro výměnu dat, který je čitelný jak člověkem tak

[3]Representational State Transfer

strojem. Je nezávislý na jakémkoli programovacím jazyce a je běžným výstupem API v nejrůznějších aplikacích. XML je značkovací jazyk, který poskytuje pravidla pro definici libovolných dat. Používá značky k rozlišení mezi atributy dat a skutečnými daty. Ačkoli se při výměně dat používají oba formáty, JSON je novější, flexibilnější a oblíbenější variantou, proto bude použit i při práci s tímto API.

Listing 2.3: JSON odpověd API Solar Edge

```
"overview":{
"lastUpdateTime":"2013-10-01␣02:37:47",
"lifeTimeData":{
"energy":761985.75,
"revenue":946.13104
},
"lastYearData":{
"energy":761985.8,
"revenue":0.0
},
API Description
22
"lastMonthData":{
"energy":492736.7,
"revenue":0.0
},
"lastDayData":{
"energy":0.0,
"revenue":0.0
},
"currentPower":{
"power":0.0
}
}
}
```

2.3.4 PV Forecast

PV Forecast[4] je aplikace na předpověď produkce fotovoltaických elektráren. Předpovídá osvit, a pak lze z výkonu elektrárny vypočítat její předpokládaný výkon do budoucnosti. V práci bude použit její výstup, který aplikace poskytuje na základě jednoduchých API dotazů, viz listing 2.4, kde je třeba zadat vygenerovaný klíč a požadované souřadnice v rámci ČR. API umožňuje volbu formátu, ve kterém bude

[4]https://wp2.pvforecast.cz/

posílat předpověď a pro práci bude zvolen formát JSON. Příklad takové odpovědi můžeme vidět v listingu 2.5.

Listing 2.4: Příklad dotazu na PV Forecast formát JSON

```
www.pvforecast.cz/api/?key=hc8217&lat=50.157&lon=14.170&format=json
```

Listing 2.5: Příklad odpovědi z PV Forecast

```
[["2016-04-12␣00:00:00",0],["2016-04-12␣01:00:00",0],["2016-04-12
02:00:00",0],["2016-04-12␣03:00:00",0],["2016-04-12␣04:00:00",0],["
    ↪ 2016-04-12
05:00:00",0],["2016-04-12␣06:00:00",0],["2016-04-12␣07:00:00",15],["
    ↪ 2016-04-12
08:00:00",80],["2016-04-12␣09:00:00",262],["2016-04-12␣10:00:00"
    ↪ ,512],["2016-
04-12␣11:00:00",538],["2016-04-12␣12:00:00",507],["2016-04-12
13:00:00",481],["2016-04-12␣14:00:00",565],["2016-04-12␣15:00:00"
    ↪ ,532],["2016-
04-12␣16:00:00",235],["2016-04-12␣17:00:00",313],["2016-04-12
18:00:00",176],["2016-04-12␣19:00:00",108],["2016-04-12␣20:00:00"
    ↪ ,28],["2016-
04-12␣21:00:00",0],["2016-04-12␣22:00:00",0],["2016-04-12␣23:00:00"
    ↪ ,0]]
```

2.3.5 Studer Web Portal

Posledním API, které bude v práci třeba použít, je rozhraní systému Studer Web Portal, který UCEEB využívá pro monitorování baterie o využitelné kapacitě 50 kWh. Sdílí téměř všechny své vlastnosti s předchozími API. Za zmínku stojí o něco horší rychlost odpovědí oproti API Mervisu a Solar Edge a také absence data logů po roce 2021, což v praxi znamená, že API a i monitorovací aplikace je v tuto chvíli použitelná pouze pro získání aktuálních, okamžitých dat. Dokumentace tohoto rozhraní [23] je vytvořena pomocí swagger ui[5].

[5] https://swagger.io/

2.4 Vizualizace informací

2.4.1 Určení vhodného kontextu

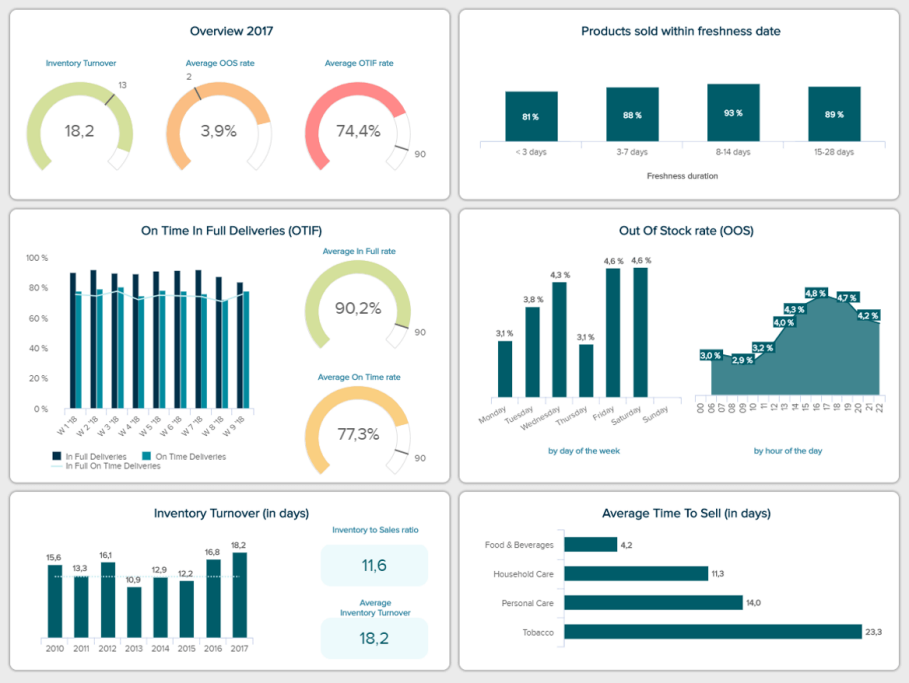
Na dashboardu lze na nejnižší úrovni rozdělení rozlišit dva typy dat - kvantitativní a kvalitativní. Vzhledem k povaze případu užití na budově se bude jednat spíše o kvantitativní data agregující číselné hodnoty. Vhodným kontextem získaných hodnot může být například jejich promítnutí v čase a vzájemné porovnání. Typickými hodnotami pro takové zobrazení jsou teplota a vlhkost. Můžeme také získat kontext tím, že hodnotu graficky porovnáme například s cílovou hodnotou nebo ji vyhodnotíme a přiřadíme k ní barvu, podobně jako například u semaforu. Příklad vhodný pro takové zobrazení je koncentrace CO_2, kde jsou přesně dané hranice únosnosti pro člověka. Kvalitativní data se v dashboardu mohou objevit například v meteorologické části, kdy můžeme dle metrik zobrazit různé obrázky popisující počasí venku – slunce, když je jasno, mrak, pokud je zataženo atp.

2.4.2 Struktura dashboardu

Pokud hovoříme o designu dashboardu, je dobré definovat termín widget, což jsou jednotlivé části rozvržení, které obvykle obsahují jednu nebo kombinaci vizuálních reprezentací, nadpis a případné metadata [24].

Pro běžné dashboardy je zde několik základních rozložení. Otevřené rozvržení umisťuje widgety (různých velikostí) způsobem bez zjevných specifických pravidel. Widgety jsou často zarovnány do mřížky. S umístěním a sousedstvím widgetů není spojena žádná silná sémantika a každý widget má stejnou nebo podobnou důležitost. Vrstvené rozložení postupuje odshora dolů a s hloubkou klesá důležitost informací. Tabulková rozložení řadí widgety do sémanticky významných sloupců a řádků. Lze je použít k opakování informací a k vizuálnímu kódování, např. napříč různými aspekty nebo datovými položkami. Rozložení tabulek usnadňuje vyhledávání a propojování informací. Seskupená rozložení seskupují dva nebo více widgetů s určitým vztahem, v mnoha případech označených společným názvem. Schematické rozvržení umisťuje widgety do nějakého schematického vztahu.

Vzhledem k povaze dat bude dashboard vyžívat kombinaci otevřeného, tabulkového a seskupeného rozvržení. Příklad takového rozvržení můžeme vidět na obrázku 2.1. Dasboard bude rozložen do dvou hlavních sekcí, a to vnitřní prostředí a energie. Obě sekce budou mít velmi podobné rozložení a to dva řádky každý po třech sloupcích, celkem 6 polí. Každé pole se bude věnovat jedné oblasti, tedy například pro vnitřní prostředí může být v jednom poli zobrazena aktuální teplota, ve vybraných místnostech a v dalším pak graf jejího průběhu. Pro energetickou část může zobrazení vypadat velmi obdobně, ale s využitím seskupeného rozvržení u widgetu zobrazující data o fotovoltaické elektrárně a widgetu grafu výkonu v čase.



Obrázek 2.1: Příklad analytického dashboardu využívajícího kombinaci otevřeného, tabulkového a seskupeného rozložení (převzato z [25]).

2.5 Konkrétní návrh uživatelského rozhraní

Na základě analýzy různých možností vizualizace informací a vhodných technologií byl vytvořen návrh uživatelského rozhraní dashboardu s využitím grafovací knihovny Plotly (viz obrázek 2.2).

Obrázek 2.2: Grafické rozhraní dashboardu zobrazující data z vnitřního prostředí a data energetická

Část II

Praktická část

Kapitola 3

Použité technologie a architektura

Tato část se věnuje technologiím a architektuře vybraných na základě analýzy. Je zde je zdůvodněno použití technologií a frameworků a následně popsána jejich implementace.

3.1 Backend

3.1.1 Logická část

Pro logickou část aplikace (backend) byl zvolen framework Flask. Hlavním důvodem je jeho existující implementace v dalších systémech na UCEEBu, která umožňuje spolupráci s ostatními programátory a dlouhodobou možnost údržby. Dalším důvodem je modularita a jednoduchost Flasku v úvodních fázích vývoje, která poměrně zásadně urychlí cestu k prvnímu prototypu, a v případě tohoto dashboardu se nebude jednat o rozsáhlou, komplikovanou aplikaci, takže je zbytečné používat těžkopádné frameworky jako je Django nebo Jakarta EE.

3.1.2 Databáze

Na základě analýzy v části 1.4.2 byla vybrána relační databáze PostgreSQL. Kromě důvodů již zmíněných v analytické části, je zde opět již existující implementace na UCEEBu.

V rámci Flasku byla pro umožnění funkce databáze použita knihovna SQLAlchemy – sada nástrojů SQL v jazyce Python, které vývojářům aplikací poskytují možnosti a flexibilitu jazyka SQL [26]. Výhodou této knihovny je, že díky funkcionalitě ORM lze databázové tabulky vytvářet a spravovat pomocí python tříd a pak s nimi dále pracovat v kódu. SQLAlchemy má zabudované funkce jako například „filter“ (viz listing 4.4), které implementují SQL funkce. Není tak nutné vkládat do python kódu SQL dotazy. Příkladem třídy a tím pádem i databázové tabulky vytvořené pomocí SQLAlchemy ORM je třída IAQSensor (viz listing 3.1), kde je zároveň vidět i tvorba databázového indexu, který umožňuje rychlejší dotazování

na data. Index je ukazatel na data v tabulce a dá se přirovnat k indexu na konci knihy. Díky jeho použití nemusí databázový engine procházet celou tabulku, když například filtruje data pomocí časového rozsahu, ale přistoupí rovnou na místo, kam ukazuje index.

Listing 3.1: Ukázka ORM mapování pomocí SQLAlchemy

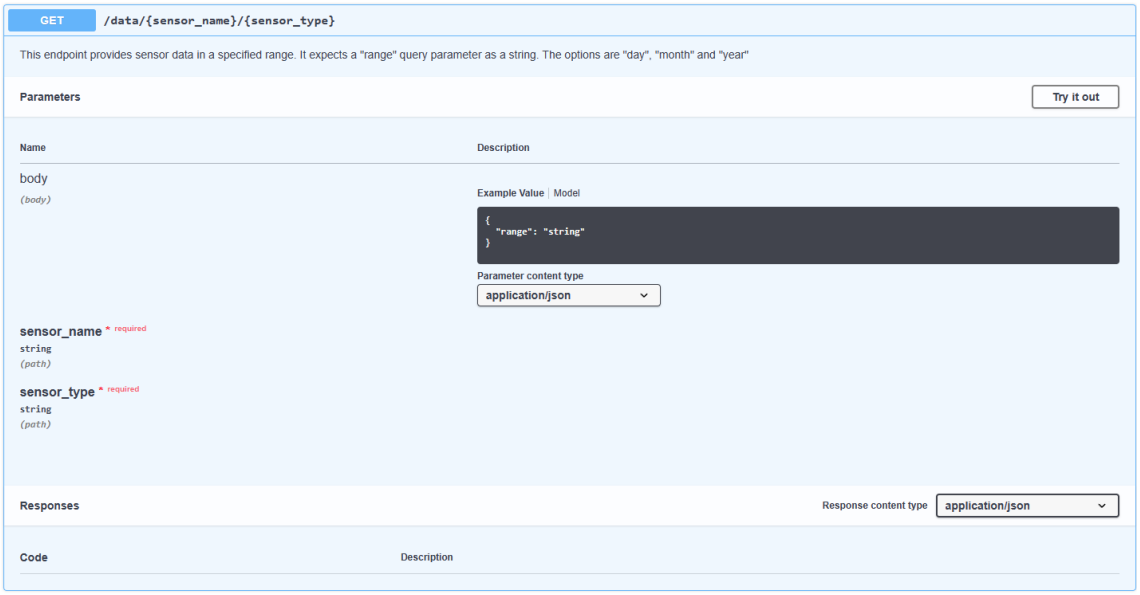
```
class IaqSensor(db.Model):
    id: Mapped[int] = mapped_column(Integer, primary_key=True)
    sensorId: Mapped[int] = mapped_column(Integer)
    deviceId: Mapped[int] = mapped_column(Integer)
    name: Mapped[str] = mapped_column(String)
    value: Mapped[Double] = mapped_column(Double)
    date: Mapped[BigInteger] = mapped_column(BigInteger)

Index('idx_sensor_device_date', IaqSensor.sensorId, IaqSensor.
    ↪ deviceId, IaqSensor.date)
```

3.2 Dokumentace

V rámci potřeby vytvořit aplikaci, kterou bude možné v budoucnu spravovat a rozšiřovat v týmu, byl kladem důraz i na dokumentaci aplikace. Kromě komentářů v samotném zdrojovém kódu byla použita i knihovna na dokumentaci rozhraní aplikace Swagger, respektive Swagger UI [27]. Swagger je platforma, která nabízí nástroje pro návrh, dokumentaci a testování API pomocí OpenAPI specifikace, což je standardní, na programovacích jazycích nezávislý popis HTTP API, který je čitelný pro člověka i počítač a umožňuje pochopit možnosti služby bez nutnosti přístupu ke zdrojovému kódu nebo další dokumentaci [28]. Jedním z klíčových komponentů Swaggeru je Swagger UI, což je vizualizační nástroj, který umožňuje vývojářům vytvářet interaktivní dokumentaci API pro prohlížení dokumentace a testování rozhraní. Swagger UI umožňuje vizualizovat API a pracovat s ním bez nutnosti použití implementační logiky. K implementaci je použita knihovna flask-apispec [29], jejíž použití je vidět na příkladu obrázku 3.2. Výsledek je pak vidět na obrázku 3.1. Tato dokumentace je přístupná na adrese, na které je dashboard momentálně nasazený s endpointem „swagger-ui“ tedy například `http://127.0.0.1:5000/swagger-ui/`.

Listing 3.2: Ukázka použití flask-apispec k tvorbě dokumentace

```
@doc(tags=['Environment'], description='This␣endpoint␣provides␣sensor
    ↪ ␣data␣in␣a␣specified␣range.␣It␣expects␣a␣"range"
query␣parameter␣as␣a␣string.␣The␣options␣are␣"day",␣"month"␣and␣"year
    ↪ "')
@use_kwargs({'range': fields.Str(required=True)})
@app.route('/data/<string:sensor_name>/<string:sensor_type>')
def get_sensor_data(sensor_name, sensor_type):
    data_range = request.args.get('range')
    return jsonify(get_sensor_data_range(sensor_name, sensor_type,
        ↪ data_range))
```



Obrázek 3.1: Ukázka vygenerované API dokumentace

3.3 Frontend

Uživatelské rozhraní je kombinací staticky vykreslovaných šablon a fetch api napsaného v javascriptu, které po načtení statických HTML elementů načítá zobrazovaná data. Pro tvorbu prostorového rozložení statické části stránky byla využita CSS[1] knihovna Bootsrap[2]. Fetch api poskytuje rozhraní pro asynchronní načítání dat. To v praxi znamená, že není třeba vždy načíst celou HTML stránku, stačí pouze zavolat dotaz získávající konkrétní data. Příkladem je použití takového dotazu při prvotním načtení stránky Vnitřní prostředí (viz listing 3.3). Poté co dojde k načtení

[1] Cascading Style Sheets
[2] https://getbootstrap.com/

základní šablony, je zavolána například tato funkce, která dále pomocí knihovny data získaná z dotazu zpracuje a vykreslí ukazatele relativní vlhkosti, teploty a koncentrace CO_2.

Listing 3.3: Ukázka použití Fetch api query

```
document.addEventListener('DOMContentLoaded', async function () {
    const response = await fetch('/data', {
        method: "GET",
        headers: {
            "Content-Type": "application/json",
        }

    });

    const body = response.json();

    createHumidity(body)
    createTemperature(body)
    createCO2(body);
});
```

Co se týká grafovací knihovny, byla vybrána knihovna Plotly. Ačkoliv je například Chart.js jednodušší na používání a rychlejší, neobsahuje takové množství grafů a možnosti jsou omezené. Z testovaných knihoven pouze Plotly vyhověla požadavkům na tuto aplikaci.

3.4 Nástroje

3.4.1 JetBrains PyCharm

Vývojové prostředí s licencí poskytnutou ČVUT. Jedná se produkt od české firmy JetBrains[3]. Toto prostředí poskytuje vše potřebné k této práci včetně podpory Flasku a tvorby uživatelského rozhraní.

3.4.2 Git

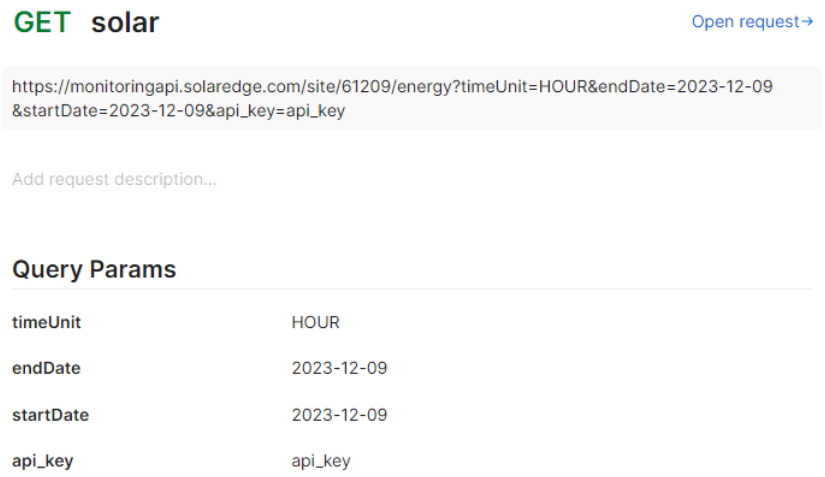
Nástroj pro ukládání a sledování změn v souborech a složkách. Git[4] sleduje každý soubor a jeho historii zvlášť. Byl použit z důvodu snadného verzování kódu a

[3]https://www.jetbrains.com/pycharm/
[4]https://git-scm.com/

možnosti ho sdílet, což zlepšuje dlouhodobou udržitelnost kódu a umožňuje týmovou spolupráci. Pro tento účel byla zvolena platforma GitLab pod hlavičkou UCEEBu.

3.4.3 Postman

Postman[5] je platforma pro testování API. Lze v něm vytvářet HTTP dotazy pro testování RESTful rozhraní. Ty pak lze dále přehledně organizovat a sdílet. V neposlední řadě Postman také umožňuje generování dokumentace. V práci byl postman využit pro testování externích API jako například na obrázku 3.2 v případě Solar Edge.



Obrázek 3.2: Ukázka dokumentace vygenerované aplikací postman

[5]https://www.postman.com/

3.5 Architektura aplikace

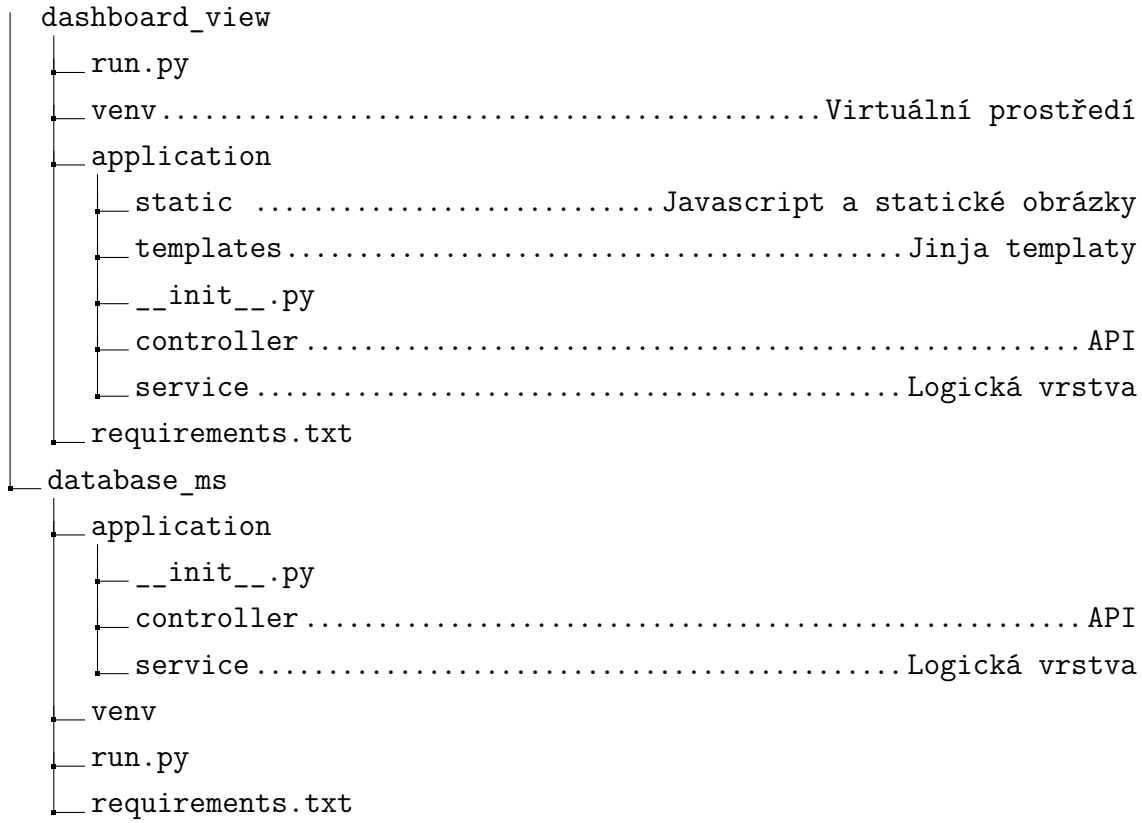
Aplikace je rozdělena do dvou celků. Jedním z nich je hlavní aplikace spravující logiku dashboardu, komunikaci s rozhraním aplikace atp. Druhým je mikroslužba zajišťující správu a aktualizaci dat v databázi. Celky mezi sebou komunikují pomocí HTTP dotazů (viz listing 3.4). V tomto konkrétním případě vidíme endpoint, na který se dotáže hlavní aplikace při prvním načtení domovské stránky. Následně je ve službě zavolána funkce „databaseSetupServices.setup_data()“, která zkontroluje databázi, a pokud je prázdná, tak do ní z externích rozhraní rozebraných v části 2.3 vloží data ze senzorů za poslední 365 dní. Pokud databáze již existuje, pouze se spustí plánovač který periodicky aktualizuje všechna relevantní data.

Listing 3.4: Ukázka endpointu databázové mikroslužby

```
@app.route('/database')
def setup_database():
    databaseSetupService.setup_data()
    global schedulerStarted
    if not schedulerStarted:
        scheduler.start()
        schedulerStarted = True
    return "Data␣query␣executed"
```

3.6 Struktura složek projektu

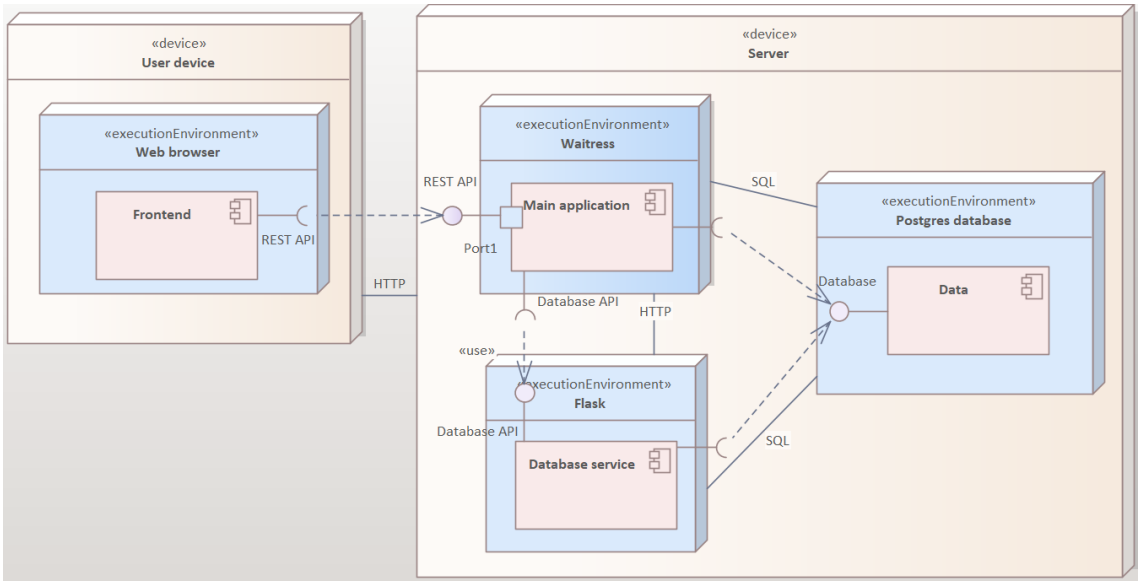
Struktura zobrazená na obrázku 3.3 se drží klasické flaskové struktury s tím rozdílem, že je zde částečně využita vrstevnatá architektura k rozdělení API vrstvy a logické vrstvy. Díky tomuto rozdělení se zlepšuje přehlednost kódu. Také je zde vidět rozdělení aplikace na hlavní a databázovou část.



Obrázek 3.3: Struktura souborů a složek

3.7 Diagram nasazení

Diagram zobrazuje komponenty systému a popisuje způsob, kterým spolu komunikují. Serverová část obsahuje dvě již zmíněné služby a Posgres databázi, kde jsou uložena data. Komunikuje přes REST API a HTTP protokol s klienty používající webové rozhraní.



Obrázek 3.4: Diagram nasazení

Kapitola 4

Design a implementace komponent dashboardu

Tato část je věnována jednotlivým komponentám/částem dashboardu. U každé je popsán jak její grafický design, tak samotná implementace v kódu. Většina komponent prošla v průběhu tvorby dashboardu poměrně výrazným vývojem, který je přiblížen v této části.

4.1 Vnitřní prostředí – identifikace okamžitých hodnot

V horní polovině obrazovky „Vnitřní prostředí“ se nachází ukazatele tří veličin – vlhkost, teplota a koncentrace CO_2 v šesti vybraných místnostech (viz obrázky 4.1, 4.2 a 4.3).

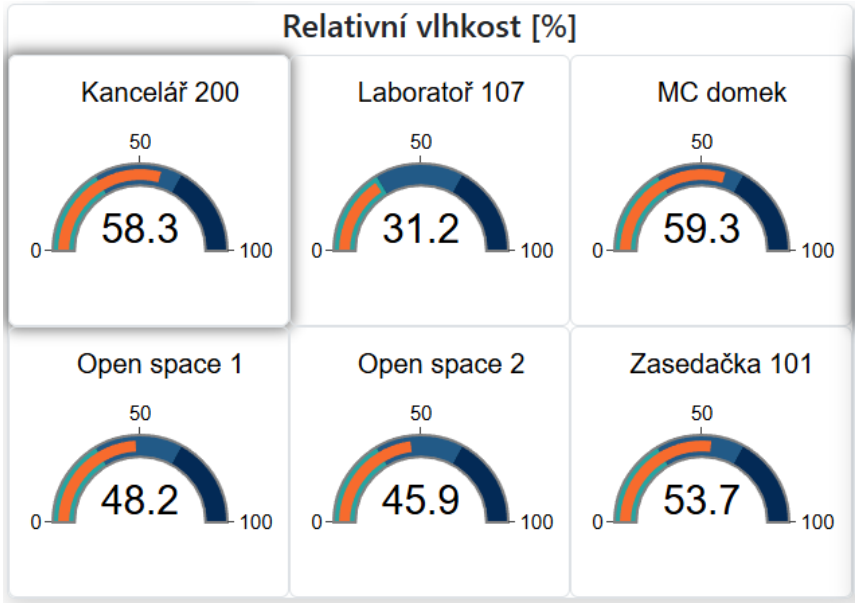
Vzhledem k tomu, že se jedná o „okamžitá“ data, byla v této části o to více zásadní rychlá čitelnost a orientace, proto je číslo identifikující hodnotu doplněno vizuálním ukazatelem.

Jeho design měl několik iterací, jednu z prvních lze vidět například na obrázku 2.2. Je vidět, že v této fázi je ukazatel obtížně čitelný, zejména není jasné, jakou hodnotu ukazuje a kde se přesně nachází „ryska“ ukazatele. Tento problém byl vyřešen menší sytostí a výrazností barev v pozadí ukazatele, který byl naopak zdůrazněn. Také zde proběhla změna barvy ukazatelů na jednotnou oranžovou a změněna barva pozadí celého grafu ze světle modré na bílou, aby byly ukazatele více kontrastní. Limitní hodnoty na ukazatelích u koncentrace CO_2 jsou zvolené na základě normy ČSN EN 16798-1 [30]. Ukazatele také slouží k výběru momentálně zobrazované místnosti v grafu, která je zvýrazněna při jeho výběru, například na obrázku 4.1 vidíme, že je zvolena místnost „Kancelář 200“, což znamená, že grafy na obrazovce zobrazují průběhy vlhkosti, teploty a koncentrace CO_2 pro tuto místnost - více v části 4.2. Tento design se ukázal jako funkční, jak bylo zjištěno v rámci uživatelského testování.

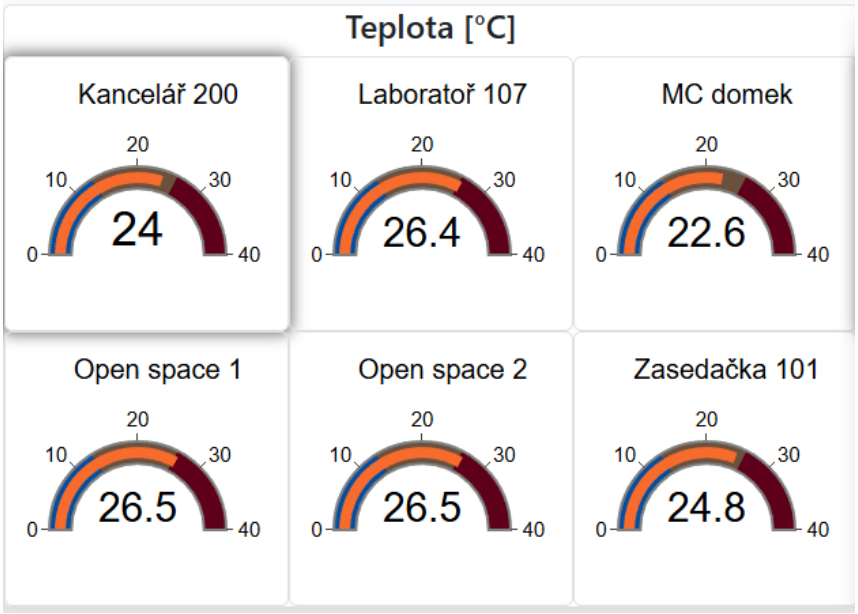
Listing 4.1: Plotly Gauge Chart

```
var data = [
  {
    domain: { x: [0, 1], y: [0, 1] },
    value: 450,
    title: { text: "Speed" },
    type: "indicator",
    mode: "gauge+number",
    delta: { reference: 400 },
    gauge: { axis: { range: [null, 500] } }
  }
];

var layout = { width: 600, height: 400 };
Plotly.newPlot('myDiv', data, layout);
```

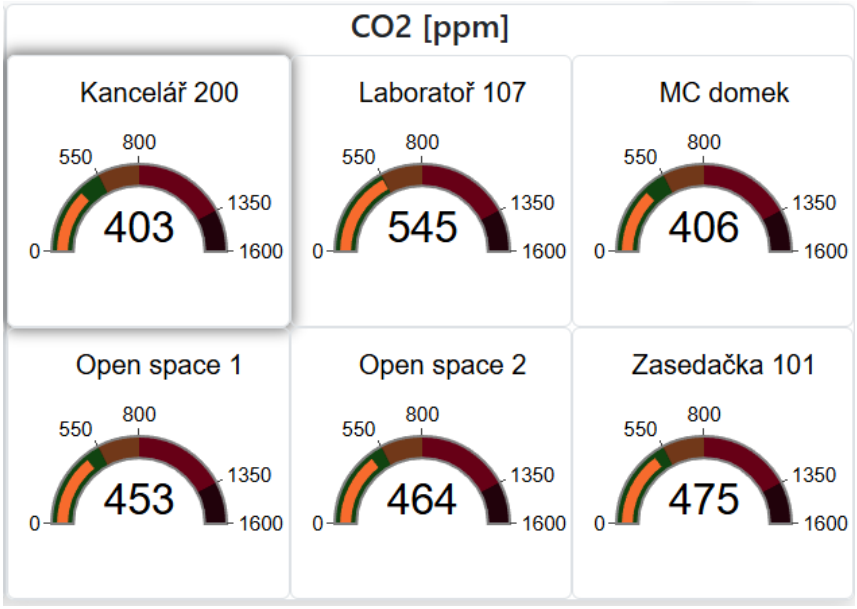
Ukazatele jsou implementovány pomocí Plotly grafu typu „indicator“, jehož jednoduchý příklad je vidět na obrázku 4.1. Tento graf je poté upraven, tak aby vyhovoval požadavkům. Jsou vykreslovány do předem připravených kontejnerů - šest pro každou veličinu a vzhledem k tomu, že se jedná o různě vypadající ukazatele, tak jejich vykreslování probíhá pomocí proměnných uložených v datové struktuře dictionary, což je neuspořádaná kolekce dat ve formátu klíč:hodnota. Funkce „setChartData“ pak přiřazuje každému typu správné vlastnosti (viz 4.2) a ukládá je do proměnné „data“ ve struktuře grafu knihovny plotly. Vstupní hodnoty „entry“ a „variable“ slouží k identifikaci hodnot v dictionary, kdy „entry“ je jednotlivá položka v datech obsahujících všechny tři veličiny a „variable“ je právě ta veličina, jejíž ukazatel funkce vytváří.



Obrázek 4.1: Ukazatele relativní vlhkosti



Obrázek 4.2: Ukazatele teploty

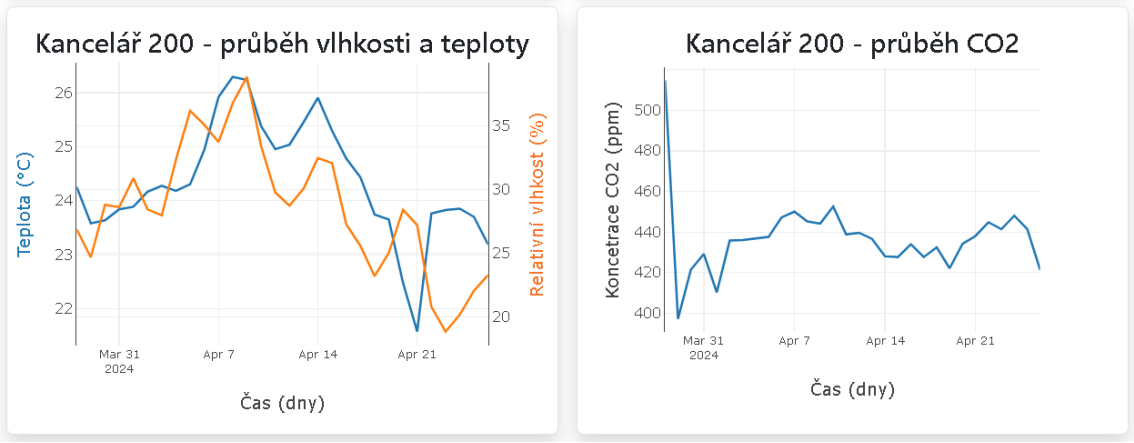


Obrázek 4.3: Ukazatele koncentrace CO_2

Bylo také třeba počítat s omezenými schopnostmi knihovny reagovat na změnu velikosti obrazovky, proto je u tohoto grafu přepočítávána velikost popisku dle velikosti obrazovky.

Listing 4.2: Funkce SetChartData

```
function setgridGraphLayout(room){
    gridGraphLayout = {
                title: {
                    text: room,
                    y: 0.9,
                    x: 0.5,
                    xanchor: 'center',
                    yanchor: 'top',
                    size: isSmallScreen ? 200 : 20 * window.innerWidth
                        ↪ / 1912,
                },
                height: isSmallScreen ? 200 : 200 * innerWidth / 1912,
                width: isSmallScreen ? 200 : 200 * innerWidth / 1912,
                margin: {
                    t: 25,
                    r: isSmallScreen ? 50 : 50 * innerWidth / 1912,
                    l: isSmallScreen ? 20 : 20 * innerWidth / 1912,
                    b: 0
                },
                paper_bgcolor: 'rgba(0,0,0,0)',
                font: { color: "black", family: "Arial", size:
                    ↪ isSmallScreen ? 15 : 15 * window.innerWidth /
                    ↪ 1912 },

            };
    return gridGraphLayout;
}
```



Obrázek 4.4: Ukázka rozhraní průběhů relativní vlhkosti, teploty a koncentrace CO_2

4.2 Vnitřní prostředí - průběh měřených hodnot

Průběh hodnot je vyobrazen v kontextu času ve spodní polovině obrazovky. Průběh odpovídá momentálně vybrané místnosti a časovému rozsahu (den, měsíc, rok). V prvním návrhu grafů se jednalo o nijak nezpracovaný průběh dat, což mělo za následek přílišnou granularitu dat v delších časových intervalech, proto bylo přistoupeno k zobrazení denních průměrů pro časové úseky měsíc a rok. Dále byla v pozdějších fázích vývoje odebrána legenda grafu teploty a vlhkosti z důvodu ušetření vertikálního prostoru a nahrazena popisky v barvě křivek. Po uživatelském testováním (kapitola 5) bylo dále zjištěno, že pro uživatele není intuitivní v jakém časovém rozsahu se graf nachází, a objevil se i problém s identifikací místnosti, kterou graf zobrazuje i přes vizuální indikaci v ukazatelích. Problém s identifikací místnosti byl vyřešen přidáním jejího názvu do nadpisu grafu. Co se týká identifikace momentálně vybraného časového rozsahu, tak bylo opět třeba tuto informaci zobrazit přímo v grafu – výběr v navigaci stránky se nachází vizuálně příliš daleko od křivek a nejedná se o intuitivní řešení. Jednou z variant bylo zobrazit podobně jako název místnosti i časový rozsah - tedy v nadpisu grafu. Toto řešení ale mělo za následek „roztažení“ nadpisu na dva řádky při zachování velikosti, a tím pádem ztrátu horizontálního prostoru pro samotný graf a opětné zhoršení čitelnosti grafu. Proto bylo přistoupeno k identifikaci pomocí měnících se jednotek v popisku osy x, kdy je pro denní rozsah zobrazen popisek „Čas (hodiny)“ a pro rok a měsíc „Čas (dny)“. Řešení není na první pohled tak jasné jako první varianta, ale znatelně pomáhá identifikovat rozsah osy x a tím i orientaci v dashboardu bez ztráty prostoru a opakování již jinde zobrazené informace. Konečná podoba křivek je vidět na obrázku 4.4, kde se nachází průběh veličin v Kanceláři 200 v časovém rozsahu měsíc.

Samotná implementace je opět za pomoci knihovny plotly.js s použitím typu „line chart“s dvěma osami y pro graf vlhkosti a teploty. Funkce vykreslující grafy jsou volány při prvotním načtení stránky, a pak také při změně vybrané místnosti nebo časového rozsahu. Načtení dat probíhá asynchronně přes fetch api, takže nedochází

k načtení celé stránky ale pouze grafů. Výběr místnosti je synchronizovaný mezi veličinami – při výběru MC Domek v komponentě „relativní vlhkost“ se ta samá místnost vybere i pro teplotu a koncentraci CO_2 a vykreslí se v obou grafech. Zmíněné denní průběhy pro delší časový rozsahy jsou realizovány pomocí databázových pohledů (views). To jsou v PostgreSQL virtuální tabulky, které reprezentují data z jedné nebo více tabulek. View je v podstatě uložený SQL dotaz, a data, která vidíme, jsou skutečná data z tabulky. Pomocí view dojde k výpočtu vždy automaticky při změně „mateřské“ tabulky a není třeba počítat průměr při každém dotazu [31]. Načítání dat je pak podstatně rychlejší. Příklad takového pohledu na tabulce pro data z vnitřního prostředí lze vidět v listingu 4.3.

Listing 4.3: Příklad databázového pohledu (view)

```
CREATE VIEW avg_iaq_sensor_value_per_day AS
SELECT
    "sensorId",
    "name",
    EXTRACT(EPOCH FROM date_trunc(
    'day', to_timestamp(date / 1000)
    )) * 1000 AS aggregated_date,
    avg(value) AS average_value
FROM
    iaq_sensor
GROUP BY
"sensorId", aggregated_date, "name";
```

4.3 Vnitřní prostředí - venkovní prostředí

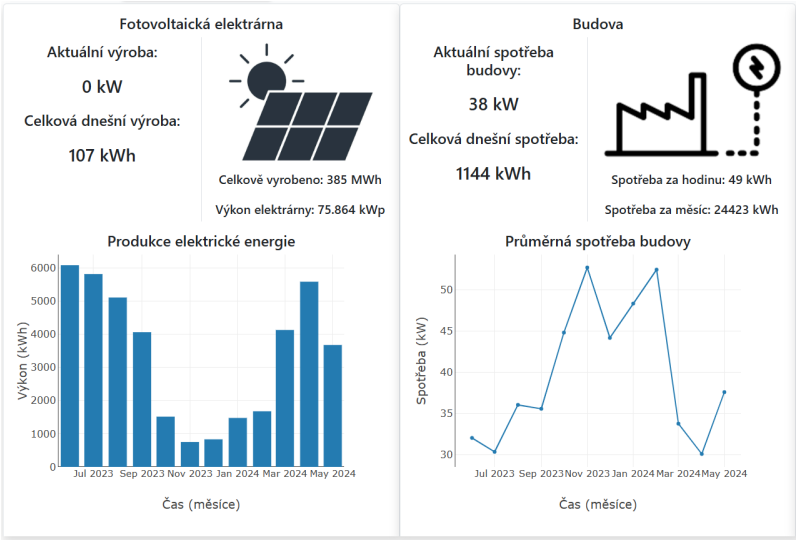
Pro poskytnutí kontextu k datům z vnitřního prostředí byl do této části dashboardu přidán prvek zobrazující data z meteorologické stanice (teplota, rychlost větru, relativní vlhkost, pravděpodobnost srážek a tlak). Tato komponenta pouze přebírá data z externího API. Na základě textové informace se vykreslují různé ikony reprezentující podmínky ve venkovním prostředí. V rámci vývoje došlo ke zvětšení písma kvůli lepší čitelnosti a v pozdější části vývoje k reorganizaci hodnot, aby se komponenta lépe přizpůsobovala velikosti obrazovky. Porovnání lze vidět na obrázku 4.5.

Obrázek 4.5: Venkovní prostředí

4.4 Energetický management - fotovoltaická elektrárna a budova

V rámci zobrazení energetických toků v budově se v této části dashboardu nachází dva bloky s textovými informacemi (aktuální příkon/výkon, statistky) a zároveň grafem viz obrázek 4.6. Původně byly bloky oddělené stejně jako na obrazovce vnitřního prostředí, v pozdějším návrhu však došlo ke spojení. Graf zobrazuje informace přímo k oblasti v horní části obrazovky a také se pak uživatelské prostředí chová lépe v případě zmenšení obrazovky, protože relevantní informace zůstanou u sebe.

Podobně jako na obrazovce vnitřního prostředí i zde původní návrh obsahoval pouze průběh nijak neagregovaných hodnot v různých časových rozsazích. Průběh absolutních hodnot s 15 minutovým intervalem v rozsahu jednoho roku neposkytuje žádnou relevantní informaci pro uživatele - data jsou hustá a orientace v grafu špatná. Proto bylo i zde přistoupeno k agregaci dat. V případě fotovoltaické elektrárny je zobrazeno celkově vyrobená energie za den - časový rozsah měsíc a za měsíc v časovém rozsahu rok. Také se mění typ grafu - v rozsazích měsíc a rok se jedná o sloupcový graf, zatímco v případě rozsahu „den“ se jedná o klasický linkový graf. V tomto rozsahu je také dostupné porovnání s předpovědí výkonu, která je stahována v podobě předpovědi osvitu přes API PV Forecast[1] a pomocí výpočtu převáděna na výkon.



Obrázek 4.6: Zobrazení informací fotovoltaické elektrárny a příkonu budovy

Při vykreslování grafů se v backendu aplikace opět pracuje s databázovými pohledy podobně jako v případě vnitřního prostředí. Jestli je dotazována mateřská tabulka nebo pohled je voleno pomocí časového rozsahu vybraného uživatelem. Funkce data vkládá do ve všech případech stejné datové struktury dictionary, aby k nim poté na frontendu šlo přistupovat stejně a nedocházelo k duplikaci kódu.

Dále byla oproti původnímu návrhu usnadněna identifikace bloků pomocí jasných nadpisů, a také byla použit intuitivnější obrázek reprezentující fotovoltaický panel.

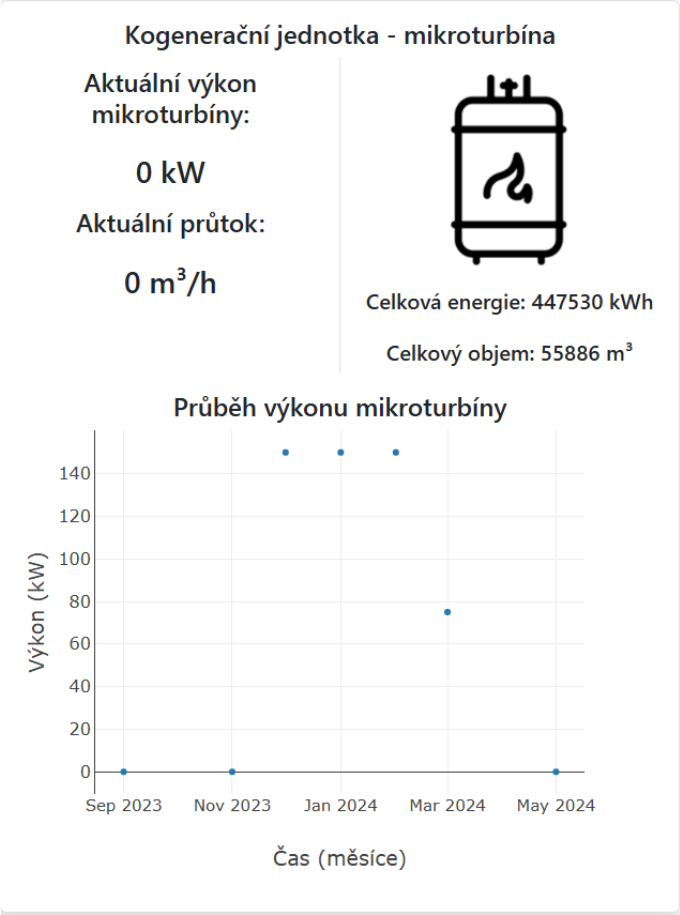
[1] https://wp2.pvforecast.cz/

Listing 4.4: Funkce dotazující se na dlouhodobá data příkonu budovy

```
def get_mervis_data_in_range(data_range):
    end_date = datetime.now()
    start_date = None

    if data_range == DAY:
        start_date = end_date - timedelta(days=1)
    elif data_range == MONTH:
        start_date = end_date - timedelta(days=30)
    elif data_range == YEAR:
        start_date = end_date - timedelta(days=365)
    else:
        raise ValueError(f"Invalid data_range: {data_range}")

    start_date_unix = int(start_date.timestamp())
    end_date_unix = int(end_date.timestamp())

    if data_range == DAY:
        results = db.session.query(MervisSensor.value, MervisSensor.
            ↪ date).filter_by(name='input_power').filter(
            MervisSensor.date.between(start_date_unix, end_date_unix))
                ↪ .order_by(
            asc(MervisSensor.date)).all()
        results_dict = [{"date": result.date, "value": result.value}
            ↪ for result in results]
    elif data_range == MONTH:
        results = db.session.query(AvgMervisSensorDay.average_value,
            ↪ AvgMervisSensorDay.aggregated_date).filter(
            AvgMervisSensorDay.aggregated_date.between(start_date_unix
                ↪ , end_date_unix)).order_by(
            asc(AvgMervisSensorDay.aggregated_date)).all()
        results_dict = [{"date": result.aggregated_date, "value":
            ↪ result.average_value} for result in results]
    else:
        results = db.session.query(AvgMervisSensorMonth.average_value
            ↪ , AvgMervisSensorMonth.aggregated_date).filter(
            AvgMervisSensorMonth.aggregated_date.between(
                ↪ start_date_unix, end_date_unix)).order_by(
            asc(AvgMervisSensorMonth.aggregated_date)).all()
        results_dict = [{"date": result.aggregated_date, "value":
            ↪ result.average_value} for result in results]

    return jsonify(results_dict)
```

4.5 Energetický management – Kogenerační jednotka, mikroturbína

V části 2.3.5 jsou zmíněna data z baterie. Komponenta pro jejich zobrazení byla vypracována (viz například obrázek 2.2), ale baterie se již nepoužívá, a proto byla nahrazena grafem zobrazujícím průběh výkonu mikroturbíny v kogenerační jednotce (viz obrázek 4.7). Data výkonu mikroturbíny získávaná z Mervis API však pravděpodobně mají špatnou jednotku, a to kW místo W. V rámci zobrazení byly hodnoty upraveny tak, aby odpovídaly výkonu ve Wattech.



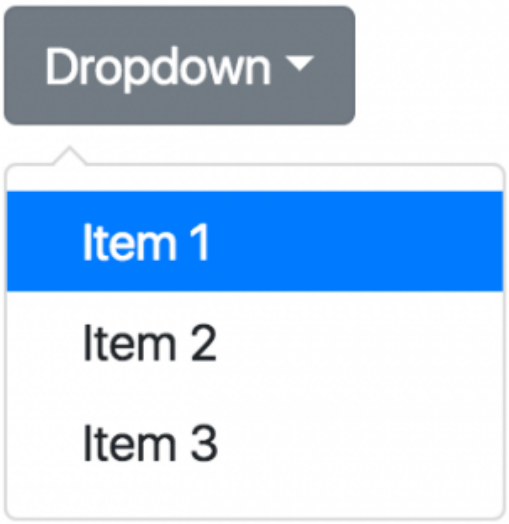
Obrázek 4.7: Kogenerační jednotka - mikroturbína

4.6 Časové zobrazení

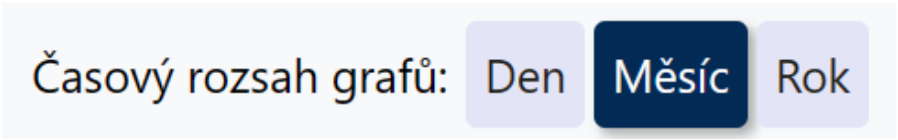
V předešlých částech kapitoly je zmíněn vývoj práce s časovým kontextem dashboardu u jednotlivých komponent – například grafů vnitřního prostředí. V této části se nachází shrnutí tohoto vývoje.

V první verzi interaktivního dashboardu bylo přepínání časových úseků řešeno pouze pomocí tzv. „drop-down“ menu viz například obrázek 4.8. Ačkoliv je toto zobrazení poměrně často používané pro výběr z možností, ukázalo se jako nevhodné

ze dvou důvodů. Za prvé je zde omezená možnost identifikace momentálně vybraného časového rozsahu a za druhé se jedná o řešení, které je často implementováno z důvodu šetření horizontálního prostoru v navigačním panelu, a to není případ této aplikace, která má navigační panel velmi jednoduchý. Z těchto důvodu byla zvolena verze zobrazující všechny možnosti v navigačním panelu s výraznou identifikací právě vybraného rozsahu viz obrázek 4.9



Obrázek 4.8: Příklad drop-down menu



Obrázek 4.9: Výběr časového rozsahu dashboardu

Po těchto změnách byl dále objeven problém s určením, k čemu se výběr vztahuje. Jedná se pouze o grafy a všechny ostatní hodnoty jsou okamžité. Tento problém byl vyřešen jednak pomocí názvu výběru rozsahu a také pomocí dynamických popisků osy x, respektive jejích jednotek (hodiny, dny, měsíce). Po změně uživatel nemusí neintuitivně hledat v hodnotách grafu, v jakém rozsahu se křivka nachází, ale může tuto informaci snadno dedukovat z popisku osy.

Kapitola 5

Testování

V rámci tvorby dashboardu proběhlo i uživatelské testování v budově UCEEBu za pomoci dotykové obrazovky. Součástí testování bylo i vyplnění krátkého dotazníku, kde měli respondenti zhodnotit práci s dashboardem. Testování probíhalo pomocí metody „First click testing“, což je metoda, která poskytuje objektivní přehled o tom, jak je webová stránka, aplikace nebo software uživatelsky přívětivý [32]. Při testu jsou účastníci požádáni, aby splnili určitý úkol, a následně se zjišťuje, jak snadné pro ně bylo tento úkol splnit. V závislosti na úkonech uživatele experiment prokáže, zda je funkce na webové stránce nebo v softwaru snadná k nalezení a použití. V tomto případě byl uživatelům prezentován scénář s 11 jednoduchými úkoly a byl jim měřen celkový čas zatímco probíhalo externí pozorování a analýza jejich akcí – například, pokud uživatelé dělali nějaké opakované chyby ve vykonávání úkolů nebo jestli v některých případech docházelo ke zmatení uživatele.

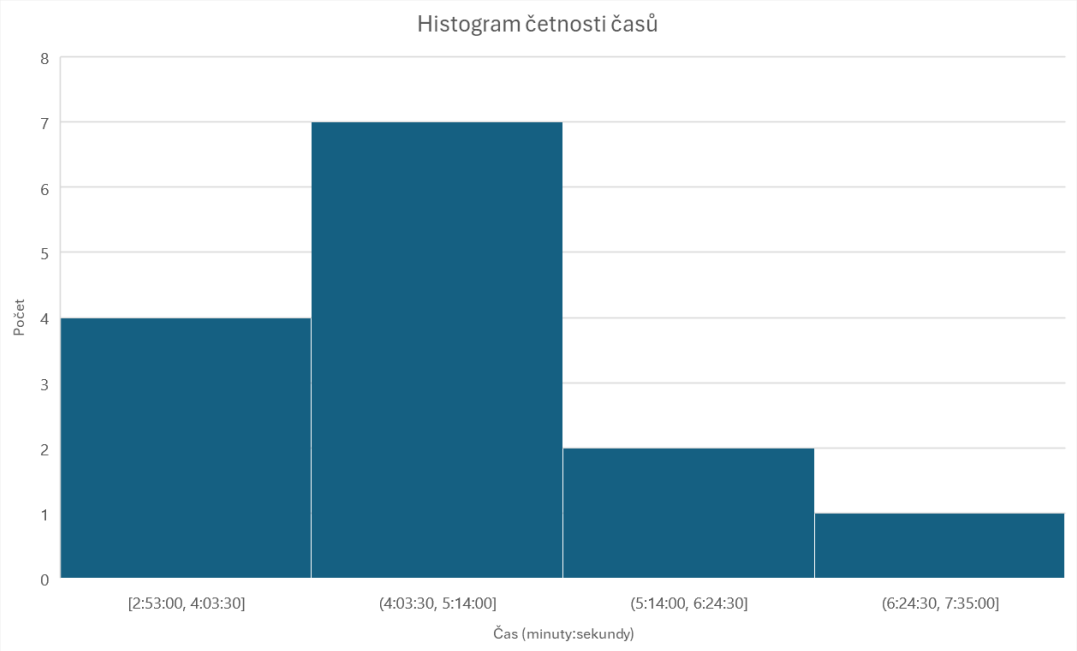
K testování byl použit následující scénář:

- TS1: Ujistěte se, že se nacházíte na obrazovce zobrazující vnitřní prostředí budovy. Pokud ne, přepněte na ni.
- TS2: Zjistěte jaká je v tuto chvíli teplota, vlhkost a koncentrace CO2 v místnosti Open space 1.
- TS3: Zobrazte denní průběh relativní vlhkosti a teploty pro místnosti MC domek a zjistěte jaká je přibližně maximální hodnota teploty za tento časový úsek.
- TS4: Zjistěte rychlost větru v exteriéru v tuto chvíli.
- TS5: Zobrazte roční průběh CO2 v místnosti Laboratoř 107. Přibližně identifikujte průměrnou hodnotu koncentrace.
- TS6: Zobrazte část dashboardu věnující se energetickému chování budovy.
- TS7: Zjistěte, v jakém časovém rozsahu jsou v tuto chvíli grafy na obrazovce.

- TS8: Zobrazte roční průběh výkonu fotovoltaické elektrárny.
- TS9: Porovnejte grafy reálného výkonu elektrárny a jeho předpovědi – v denním rozsahu.
- TS10: Zjistěte, jestli v tuto chvíli budova vyrábí více energie než spotřebovává. Pokud ano, kolik. Dále zjistěte, jaký je v tuto chvíli příkon budovy.
- TS11: Zjistěte, jaký je aktuální výkon mikroturbíny a kolik energie vyrobila celkově.

5.1 Výsledky testování

Uživatelského testování se zúčastnilo celkem 14 respondentů. V grafu na obrázku 5.1 je vidět, jak si uživatelé vedli z hlediska času při vykonávání scénáře. Jedná se v podstatě o normální distribuci s tím, že většina časů nepřesáhla pět minut a průměrný čas byl 4 minuty a 36 sekund.



Obrázek 5.1: Histogram

Většina uživatelů neměla s užíváním dashboardu zásadní problémy, přesto se však objevilo několik opakujících se nedostatků. Jedním z nich byl problém s přepnutím do denního rozsahu grafů v úkolu TS3. Uživatel úspěšně zobrazil průběh relativní vlhkosti a teploty pro místnosti MC domek ale v měsíčním rozsahu. Naproti tomu v úkolu TS5 tento problém uživatelé neměli. Jednalo se pravděpodobně i o formulaci úkolu, je však jasné, že přepínání grafů nebylo intuitivní. Uživatelé často hledali a nebylo jasné, k čemu se vlastně přepínání vztahuje. Také na dashboardu nebyla žádná identifikace přímo u grafu, která by napomohla k identifikaci daného

rozsahu. Jeden z respondentů měl tento problém i s výběrem místnosti, která je zobrazena v grafu.

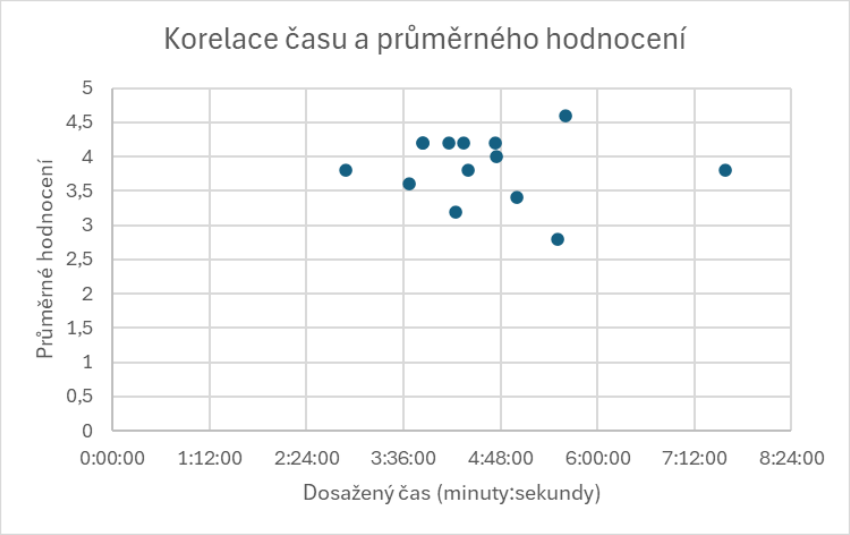
V energetické části dashboardu byl u úkolu TS8 často problém s identifikací komponenty fotovoltaické elektrárny. Na obrazovce „Energie“ okno nalevo nemělo kromě ikony panelu žádnou další identifikaci. Problém s tímto spojený byl u úkolu TS:9, kdy jeden z uživatelů nedokázal zobrazit relevantní kontext a porovnat předpověď s reálnou výrobou. Nakonec u úkolu TS10 jeden uživatel narazil na problém s identifikací termínu „příkon“, který není pro širokou veřejnost stoprocentně přívětivý. Na druhou stranu s úkoly 1, 2, 5, 6 a 7 neměl žádný z testovaných uživatelů výrazný problém.

V rámci respondentů byli jak uživatelé, kteří se věnují problematice energetiky, senzoriky, energetického monitoringu nebo prezentaci dat, tak i uživatelé, kteří se na těmto oblastem nevěnují vůbec. Problémy měly obě skupiny velmi podobné. V případě zasvěcených uživatelů však nedocházelo k problémům popsaných v předchozím odstavci k energetické části dashboardu – identifikace informací byla z jejich strany bezproblémová. To se pravděpodobně odrazilo i na průměrném čase, kdy tato skupina měla průměrný čas 4 minuty 19 sekund oproti 4 minutám a 54 sekundám druhé skupiny.

5.2 Výsledky šetření

Po absolvovaní testovacího scénáře byli uživatelé požádáni o vyplnění krátkého dotazníku s hodnocením 1 až 5, kdy hodnocení 1 znamená rozhodný nesouhlas s tvrzením v otázce a 5 rozhodný souhlas.

Než bude přistoupeno k rozboru jednotlivých otázek, nabízí se ověřit hypotézu, jestli v rámci testování a následného šetření vznikla významná korelace mezi dosaženým časem a průměrným hodnocením dashboardu jednotlivých uživatelů, tedy jestli delší čas průchodu scénářem ovlivnil uživatele natolik, že dashboard pak hodnotil negativně. Na grafu v obrázku 5.2 lze vidět závislost průměrného hodnocení na dosaženém čase. Korelační koeficient mezi těmito dvěma množinami je -0.27, což v tomto případě není statisticky významná korelace a čas průchodu testovacím scénářem neovlivnil následné hodnocení v dotazníku.



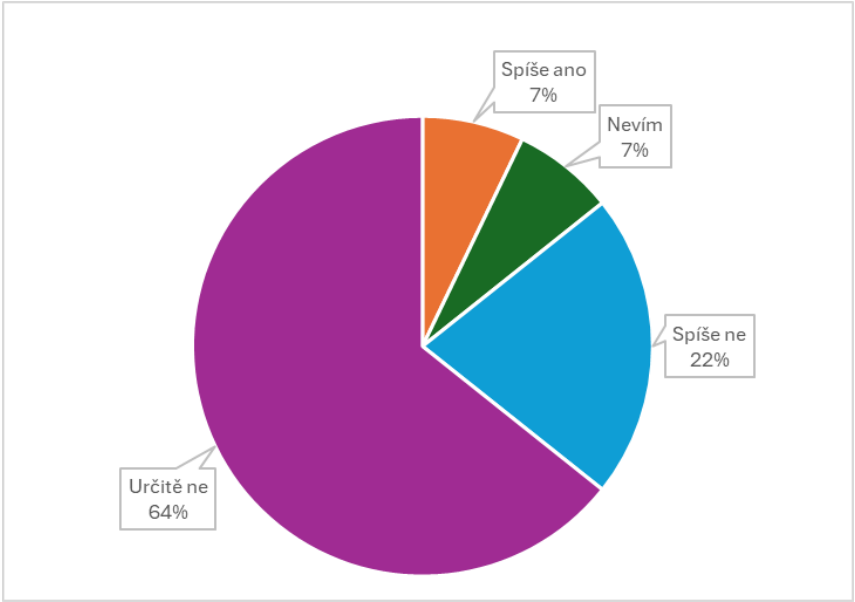
Obrázek 5.2: Korelace mezi časem a hodnocením

5.2.1 Otázka 1: Dashboard je příliš komplikovaný

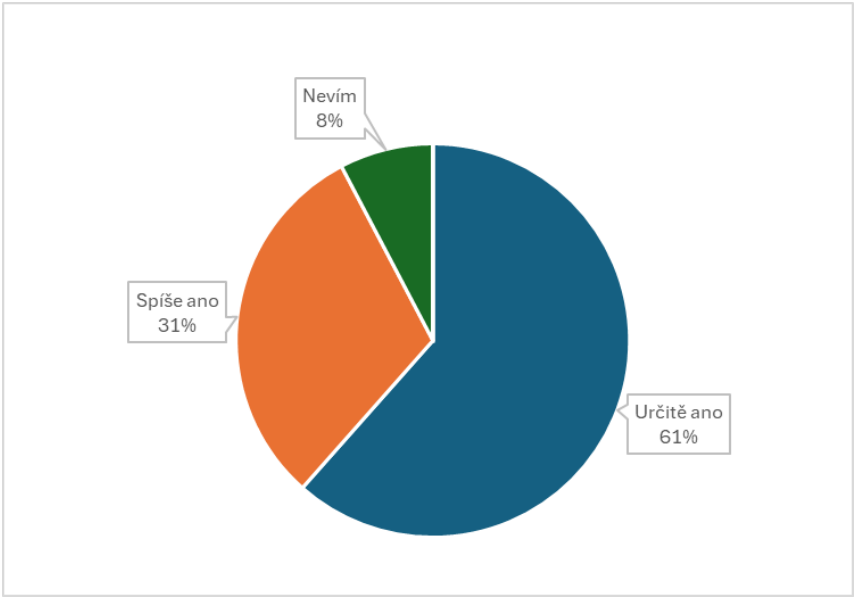
Cílem této otázky bylo zjistit, jak na uživatele dashboard celkově působí. Nesouhlas vyjádřilo 86 procent respondentů, 7 procent zvolilo prostřední neutrální možnost a 7 procent se vyjádřilo souhlasně, tedy že dashboard je příliš komplikovaný. Pro celková statistická šetření byla v této otázce provedena inverze hodnocení, aby byla v souladu s ostatními.

5.2.2 Otázka 2: Používání dashboardu je intuitivní

Otázka měla za cíl zjistit, jak se uživatelům pracovalo s dashboardem a jestli měli při práci nějaké problémy s vykonáním zadání. Zde vyjádřilo souhlas celkově 92 procent respondentů a 8 procent se přiklonilo k neutrální možnosti.



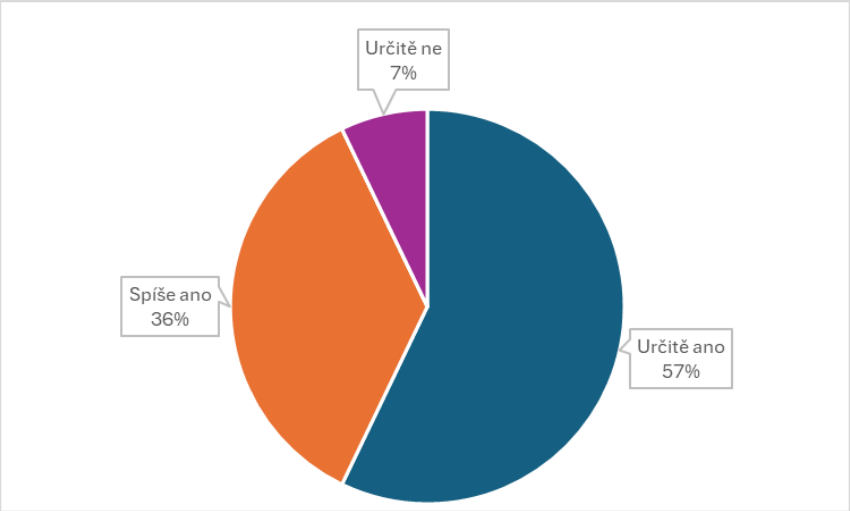
Obrázek 5.3: Otázka 1: Dashboard je příliš komplikovaný



Obrázek 5.4: Otázka 2: Používání dashboardu je intuitivní

5.2.3 Otázka 3: Uživatelské prostředí dashboardu je konzistentní

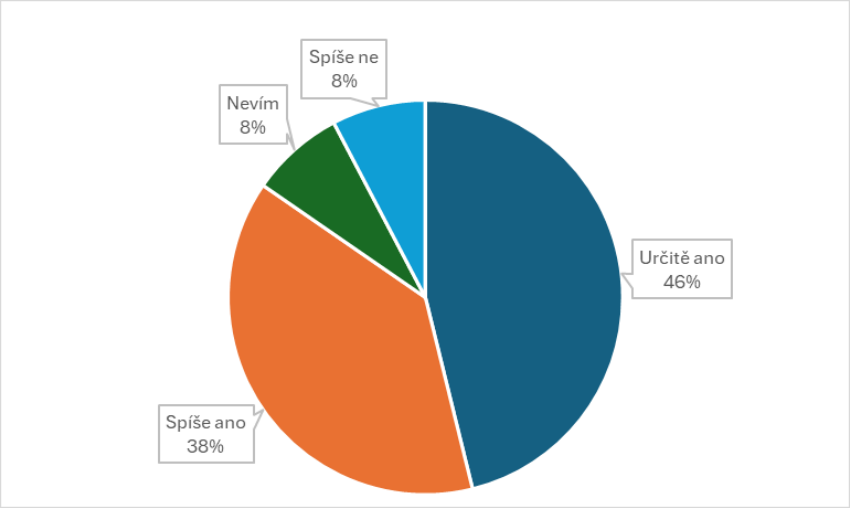
Cílem této otázky bylo zjištění, jestli je uživatelům dashboard příjemný z hlediska grafické konzistence tedy, jestli obrazovky drží nějaký styl/vzor, dle kterého se lze orientovat. Zde opět výrazná většina (93 procent) uvedla, že uživatelské prostředí konzistentní je. 7 procent respondentů naopak vyjádřilo silný nesouhlas. Výsledky této otázky mohou být ovlivněny i jejím vnímáním, protože v případě, že respondent hodnotil konzistenci vždy jedné z obrazovek se výsledky pravděpodobně budou lišit od případu, že uživatel hodnotil dashboard jako celek.



Obrázek 5.5: Otázka 3: Uživatelské prostředí dashboardu je konzistentní

5.2.4 Otázka 4: Kontext zobrazení dat je pochopitelný a intuitivní

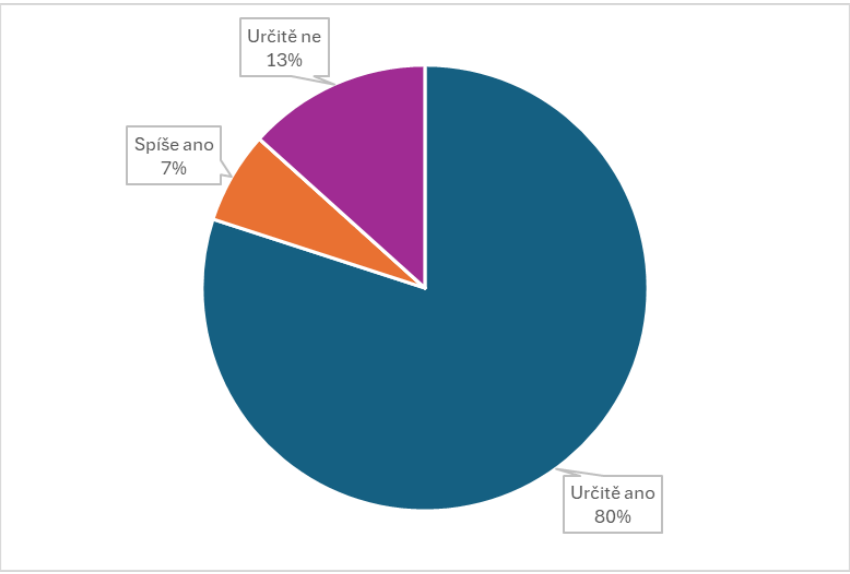
Zde bylo cílem zjistit, jak uživatel/respondent vnímá kontext zobrazení dat, tedy zejména názvy ukazatelů, grafů a komponent, a také byla zaměřena na práci s časovým kontextem dat a jeho výběrem. Ačkoliv byly výsledky mírně „horší“ oproti ostatním otázkám, i tak vyjádřilo celkově 84 procent respondentů souhlas s tvrzením. 8 procent se přiklonilo k neutrální odpovědi a 8 procent vyjádřilo mírný nesouhlas. Zde je zajímavé, jak již bylo zmíněno v části 5.1 pojednávající o výsledcích testování, že s touto částí (identifikace kontextu zobrazení dat) měli uživatelé největší problém. Pravděpodobně však zmatení na jednom z bodů testovacího scénáře nezpůsobilo výraznější ovlivnění celkového dojmu z uživatelského prostředí.



Obrázek 5.6: Otázka 4: Kontext zobrazení dat je intuitivní a pochopitelný

5.2.5 Otázka 5: Textové informace jako popisky os a okamžitých hodnot jsou dobře čitelné

Podobně jako u předchozí otázky bylo cílem analyzovat, jak uživatel vnímá kontext zobrazení dat, v tomto případě byla otázka zaměřena konkrétně na textové informace a popisky os. V kontextu testování se jedná o nejhůře hodnocenou otázku s 14 procenty respondentů vyjadřujících silný nesouhlas. I tak ale celkově 87 procent respondentů vyjádřilo souhlas s tvrzením.



Obrázek 5.7: Otázka 5: Textové informace jako popisky os a okamžitých hodnot jsou dobře čitelné

5.2.6 Otázka 6: Otevřená otázka – návrhy, připomínky

- Odpověd 1: Výrazněji bych označil část věnující se fotovoltaice. U vnitřního prostředí není úplně jasné, že se rozsah mění v menu nahoře.

- Odpověd 2: Zoomování časových rozsahů je nepřátelské, přepíná rozsahu rok/měsíc/den je moc nahoře.

- Odpověd 3: Ikona fotovoltaiky by měla být jemnější, možná lepší ikona celkově. V záložce Energie bych volil jednotný nadpis jednotlivých boxů (FV, příkon, baterie, mikroturbína).

Odpovědi na tuto otázku potvrzují poznatky z testování v oblasti identifikace časového rozsahu a jeho přepínání. Dále zmiňují hodnotné poznatky v energetické části dashboardu co se týká přehlednost a čitelnosti jednotlivých bloků informací, a k čemu se vztahují.

5.3 Provedené úpravy

Ačkoliv je v kapitole 4 rozebrán postup tvorby jednotlivých komponent, je nutné zmínit z celkového pohledu změny provedené na základě uživatelského testování.

Na obrazovce vnitřního prostředí byl vyřešen problém s identifikací časového rozsahu pomocí dynamických popisů osy x u křivek. Dále byla opakovaně zmíněna nejasnost ohledně toho, co vlastně výběr časového rozsahu ovlivňuje – bylo tedy v jeho názvu upřesněno, že se jedná o grafy. Pro lepší identifikaci vybrané místnosti byl přidán její název do nadpisu grafů. Na obrázku 5.8 vidíme například vybranou Kancelář 200, jejíž jméno se nachází i u grafu a jako časový rozsah grafů je zvolen měsíc, a tím pádem je popisek osy x „Čas (den)“. Také došlo z důvodu lepší čitelnosti ke zvětšení písma v pravé části widgetu venkovního prostředí. Předchozí implemetace měla poměrně malé písmo a přílišné množství prázdného prostoru.

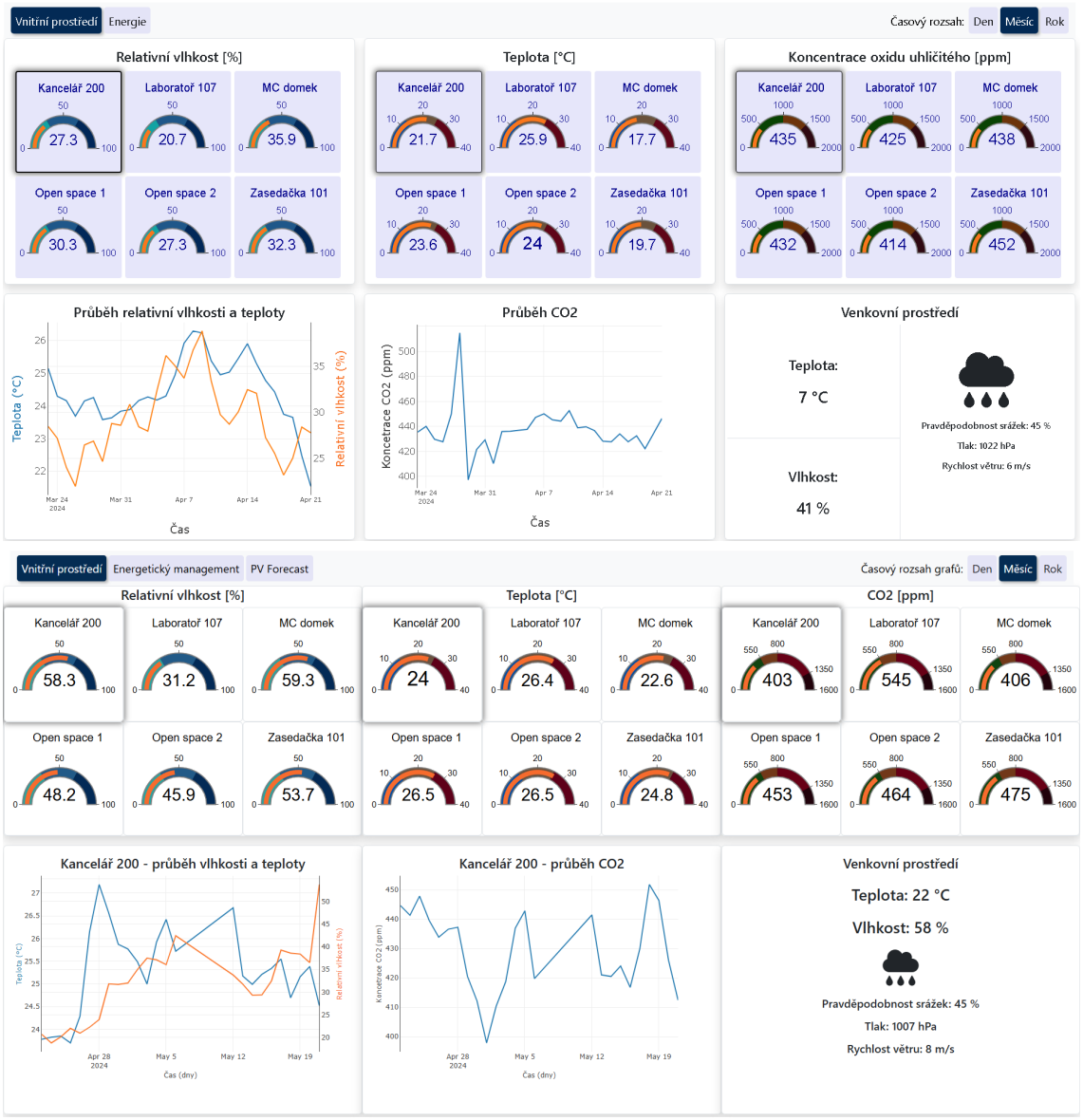
Na obrazovce energického managementu došlo v první řadě k jejímu přejmenovaní z „Energie“ právě na „Energetický management“. Tento termín mnohem lépe vystihuje podstatu obrazovky a poskytuje lepší představu o tom, co se na ní nachází. Dále byla na základě opakovaných verbálních vstupů od uživatelů i problémů ve vykonávaní relevantních úkolu ve scénáři zlepšena identifikace bloků pomocí názvů „Fotovoltaická elektrárna“ a „Budova“. Podobně jako v případě obrazovky vnitřního prostředí jsou implementovány dynamické popisky os s tím, že je zde navíc ještě pohled pro časový rozsah rok, kdy se název osy x u obou grafů změní na „Čas (měsíc)“. Nakonec byla změněna ikona reprezentující panel fotovoltaické elektrárny, protože v původní verzi nebylo jasné, že se jedné o panel.

Dále byla na podnět jednoho z uživatelů přidána třetí obrazovka PV Forecast[1]. Jedná se pouze a přímou reprodukci výstupu ze stejnojmenných stránek zobrazujících předpověď osvitu pro Českou republiku viz obrázek 5.10.

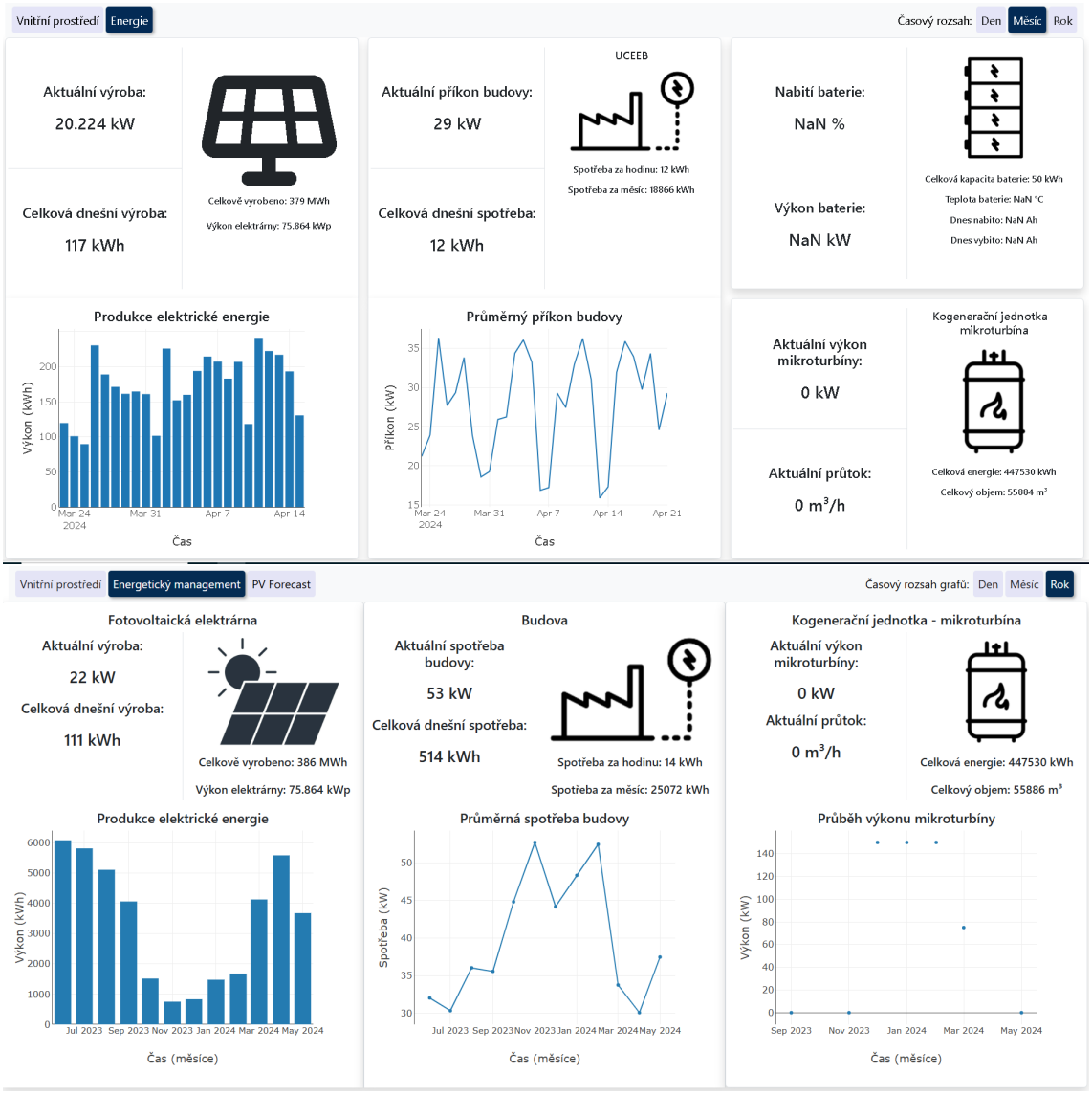
5.4 Testování v prohlížeči

Po uživatelském testování, které probíhalo na velké obrazovce o rozlišení 4K, bylo přistoupeno k testování ve webovém prohlížeči. Ukázalo se, že aplikace se chová nepředvídatelně (například při zmenšení obrazovky). Informace se stávají nečitelnými, nebo není kompatibilní s rozlišením QuadHD, které je v dnešní době poměrně běžné. Bylo tedy třeba zajistit alespoň základní úroveň responsivního chování, což proběhlo pomocí knihovny Boostrap a dynamického přepočítávaní velikosti grafů dle velikosti obrazovky. V důsledku této implementace došlo i z vizuálního hlediska k několika změnám, a to reorganizace komponent „Venkovní prostředí“ a „Kogenerační jednotka – mikroturbína“. Výslednou podobu dashboardu a porovnání s verzí před uživatelským testováním lze pozorovat na obrázcích 5.8 a 5.9.

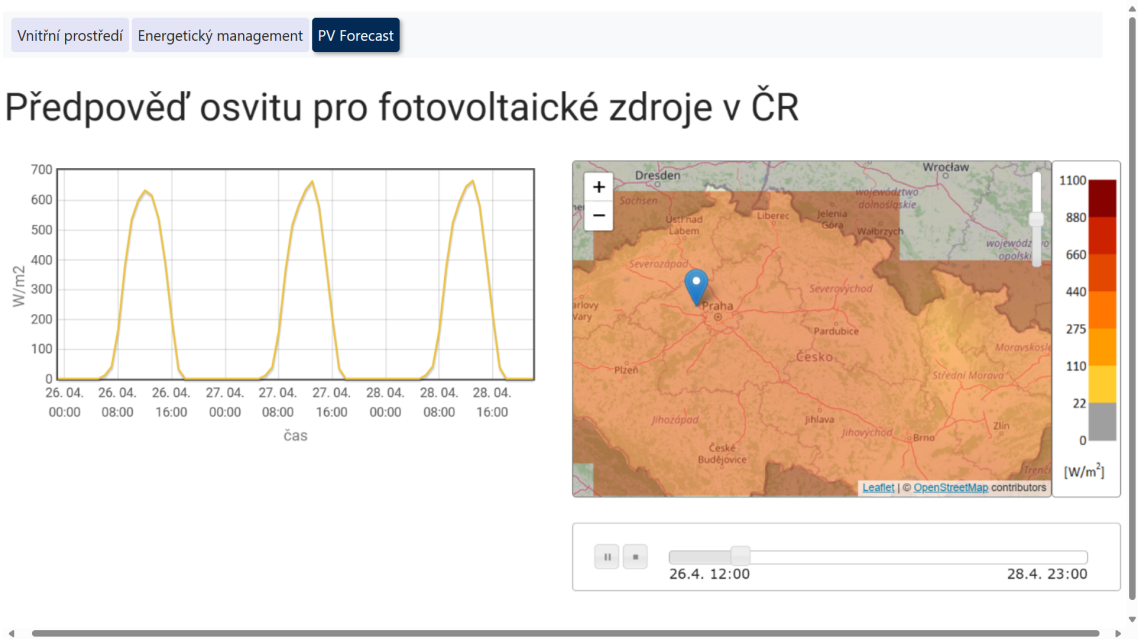
[1] https://wp2.pvforecast.cz/predpoved-osvitu/



Obrázek 5.8: Obrazovka zobrazující vnitřní prostředí před (nahoře) a po (dole) testování



Obrázek 5.9: Obrazovka zobrazující energetický management budovy před (nahoře) a po (dole) testování



Obrázek 5.10: Obrazovka předpovědi osvitu pro Českou republiku

Závěr

Zadáním práce bylo vytvořit interaktivní dashboard pro budovu UCEEB, který bude přehlednou formou prezentovat energetické a jiné ukazatele návštěvníkům z řad laické veřejnosti. Důraz měl být kladen na uživatelskou přívětivost, snadné ovládání a přehlednost zobrazených dat. Součástí zadání bylo také uživatelské testování. Toto zadání jsem splnil. Dashboard funguje a je instalován na serveru UCEEB.

V první fázi práce jsem provedl analýzu požadavků na aplikaci dostupných technologií a softwarových architektur. Zanalyzoval jsem externí rozhraní pro získávání dat a popsal postup práce s nimi. V této části jsem také vytvořil první grafický návrh, který jsem poté postupně iteroval a vylepšoval.

V druhé části jsem pak podrobně popsal konkrétní implementaci jak z technologického hlediska, tak z hlediska samotného iteračního vývoje zejména uživatelského rozhraní aplikace. Také jsem zorganizoval uživatelské testování aplikace na UCEEBu a na jeho základě provedl řadu změn.

Na této práci jsem si také vyzkoušel nové technologie, především Flask a obecně vývoj uživatelského rozhraní aplikace a nástrojů s tím spojených.

Závěrem lze říci, že tato práce přispívá do oblasti technologií inteligentních budov, protože poskytuje praktické řešení intuitivního a uživatelsky přívětivého informačního dashboardu. Dashboard, určený především pro návštěvníky budovy, efektivně vizualizuje údaje ze senzorů v jednoduchém a čitelném formátu. Důraz na uživatelský komfort, čitelnost a snadné použití vedl k vytvoření nástroje, který nejenže zlepšuje pochopení provozních parametrů budovy ze strany návštěvníků, ale také obohacuje jejich celkový zážitek. Budoucí práce by se mohla zaměřit na zpětnou vazbu od uživatelů a iterační procesy návrhu, aby bylo možné dále zdokonalit uživatelské rozhraní. Také je zde možnost přidání dalších variant zobrazení dat, mezi kterými by mohl uživatel přepínat – například jiný typ nebo kontext grafů.

Bibliografie

1. RASMUSSEN, Nils H.; BANSAL, Manish; CHEN, Claire Y. *Business Dashboards: A Visual Catalog for Design and Deployment*. John Wiley & Sons, Incorporated, 2009.
2. RICHARDS, Mark. *Software Architecture Patterns*. Sv. 4. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Incorporated, 2015.
3. LEFF, A.; RAYFIELD, J.T. Web-application development using the Model/-View/Controller design pattern. In: *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*. 2001, s. 118–127. Dostupné z DOI: 10.1109/EDOC.2001.950428.
4. DOCS, MDN Web. *MVC - MDN Web Docs Glossary: Definitions of Web-related terms* [online]. Mozilla, 2023. [cit. 2024-01-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/MVC.
5. TEAM, Flask Documentation. *Welcome to Flask — Flask Documentation (3.0.x)* [online]. Pallets Projects, 2023. [cit. 2024-01-04]. Dostupné z: https://flask.palletsprojects.com/.
6. DWYER, Gareth. *Flask By Example* [online]. Birmingham, UK: Packt Publishing, 2016 [cit. 2024-01-04]. Dostupné z: https://www.amazon.com/Flask-Example-Gareth-Dwyer/dp/1785286935.
7. LIAWATIMENA, Suryadiputra; HENDRIC SPITS WARNARS, Harco Leslie; TRISETYARSO, Agung; ABDURAHMAN, Edi; SOEWITO, Benfano; WIBOWO, Antoni; GAOL, Ford Lumban; ABBAS, Bahtiar Saleh. Django Web Framework Software Metrics Measurement Using Radon and Pylint. In: *2018 Indonesian Association for Pattern Recognition International Conference (INAPR)*. 2018, s. 218–222. Dostupné z DOI: 10.1109/INAPR.2018.8627009.
8. DREAMS, Robot. *Spring Framework vs Jakarta EE* [online]. Robot Dreams, 2023. [cit. 2024-01-04]. Dostupné z: https://robotdreams.cz/blog/167-spring-framework-vs-jakarta-ee.
9. GEEKSFORGEEKS. Prototyping Model - Software Engineering. 2024. Dostupné také z: https://www.geeksforgeeks.org/software-engineering-prototyping-model/.

10. VISWANATHAN, Viswa. Rapid Web Application Development: A Ruby on Rails Tutorial. *IEEE Software*. 2008, roč. 25, č. 6, s. 98–106. Dostupné z DOI: 10.1109/MS.2008.156.

11. JATANA, Nishtha; PURI, Sahil; AHUJA, Mehak; KATHURIA, Ishita; GOSAIN, Dishant. A survey and comparison of relational and non-relational database. *International Journal of Engineering Research & Technology*. 2012, roč. 1, č. 6, s. 1–5.

12. TAYLOR, Dan. *Client-Side Vs. Server-Side Rendering - Search Engine Journal* [online]. Search Engine Journal, 2023. [cit. 2024-01-04]. Dostupné z: https://www.searchenginejournal.com/client-side-vs-server-side/482574/.

13. VELAVAN, Kavin. *React Vs Flask* [online]. Medium, 2023. [cit. 2024-01-04]. Dostupné z: https://medium.com/@gkavinvelavan/react-vs-flask-957c2799b0f7.

14. FUSIONCHARTS. *FusionCharts Dev Centre* [online]. 2024. [cit. 2024-05-12]. Dostupné z: https://www.fusioncharts.com/dev/fusioncharts. Accessed: May 12, 2024.

15. *Chart.js Documentation*. Chart.js, 2024. Dostupné také z: https://www.chartjs.org/docs/latest/.

16. *Plotly JavaScript Documentation*. Plotly, 2024. Dostupné také z: https://plotly.com/javascript/.

17. UCEEB. *Budova UCEEB* [online]. [cit. 2024-01-02]. Dostupné z: https://www.uceeb.cz/cz/budova-uceeb/.

18. VOJÁČEK, Antonín. *Kvalita vzduchu v uzavřených místnostech - VOC index* [online]. HW, 2020. [cit. 2024-01-04]. Dostupné z: https://automatizace.hw.cz/kvalita-vzduchu-v-uzavrenych-mistnostech-voc-index.html.

19. *IAQ Indoor Air Quality Senzory*. University Center for Energy Efficient Buildings (UCEEB), 2024. Dostupné také z: https://www.uceeb.cz/cz/iaq-indoor-air-quality-senzory/.

20. *The World of Mervis - Knowledge Base API*. Energocentrum Plus, s.r.o., 2024. Dostupné také z: https://kb.mervis.info/doku.php/en:mervis-scada:50-api.

21. SOLAREDGE. *SolarEdge Monitoring API*. 2024. Dostupné také z: https://knowledge-center.solaredge.com/sites/kc/files/se_monitoring_api.pdf.

22. GEEKSFORGEEKS. *Difference between JSON and XML* [online]. 2023. [cit. 2023-12-28]. Dostupné z: https://www.geeksforgeeks.org/difference-between-json-and-xml/.

23. STUDER. *Studer Web API*. 2024. Dostupné také z: https://api.studer-innotec.com/swagger/ui/index.

24. BACH, Benjamin; FREEMAN, Euan; ABDUL-RAHMAN, Alfie; TURKAY, Cagatay; KHAN, Saiful; FAN, Yulei; CHEN, Min. Dashboard Design Patterns. *IEEE Transactions on Visualization and Computer Graphics*. 2023, roč. 29, č. 1, s. 342–352. Dostupné z DOI: 10.1109/TVCG.2022.3209448.

25. CALZON, Bernardita. *What Is A Data Dashboard? Definition, Meaning & Examples* [online]. datapine, 2023. [cit. 2024-01-04]. Dostupné z: https://www.datapine.com/blog/data-dashboards-definition-examples-templates/.

26. SQLALCHEMY. *The Python SQL Toolkit and Object Relational Mapper* [online]. SQLalchemy, 2024. [cit. 2024-01-28]. Dostupné z: https://www.sqlalchemy.org/.

27. SWAGGER. *Swagger Documentation* [online]. 2024. [cit. 2024-05-10]. Dostupné z: https://swagger.io/docs/.

28. INITIATIVE, OpenAPI. *OpenAPI Specification v3.1.0* [online]. 2021. [cit. 2024-05-10]. Dostupné z: https://spec.openapis.org/oas/latest.html.

29. CONTRIBUTORS, Flask-apispec. *flask-apispec: Auto-documenting REST APIs for Flask* [online]. 2024. [cit. 2024-05-12]. Dostupné z: https://flask-apispec.readthedocs.io/en/latest/usage.html.

30. *ČSN EN 16798-1*. Český normalizační institut, 2020. Dostupné také z: https://www.technicke-normy-csn.cz/csn-en-16798-1-127027-166817.html#.

31. POSTGRESQL. *CREATE VIEW* [online]. 2024. [cit. 2024-04-26]. Dostupné z: https://www.postgresql.org/docs/current/sql-createview.html. Dokumentace PostgreSQL 16.2.

32. FONTANELLA, Clint. *A Beginner's Guide to First Click Testing* [online]. HubSpot Blog, 2022 [cit. 2024-04-24]. Dostupné z: https://blog.hubspot.com/service/click-test.