FAKULTA STROJNÍ TUL

Diplomová práce

METEOROLOGICKÁ STANICE SE ZÁZNAMEM DAT

Studijní program: N0788A270004 Inovační a průmyslové inženýrství

Autor práce: **Bc. Radek Tvrzník**

Vedoucí práce: Ing. Michal Moučka, Ph.D.
Katedra výrobních systémů a automatizace

Liberec 2023

FAKULTA STROJNÍ TUL

Zadání diplomové práce

METEOROLOGICKÁ STANICE SE ZÁZNAMEM DAT

Jméno a příjmení:	**Bc. Radek Tvrzník**
Osobní číslo:	S22000268
Studijní program:	N0788A270004 Inovační a průmyslové inženýrství
Zadávající katedra:	Katedra výrobních systémů a automatizace
Akademický rok:	2023/2024

Zásady pro vypracování:

1. Proveďte rešerši 32bitových mikrokontrolérů, které nabízí současný trh (STM32, ESP32, PIC32MZ atd.). Prozkoumejte dostupné databázové systémy (MySQL, PostgreeSQL, IBM DB2 apod.).
2. Navrhněte strukturu meteostanice se záznamem dat a online vizualizací časových průběhů environmentálních veličin.
3. Na základě rešerše vyberte vhodný databázový systém a vývojovou desku s vybraným 32bitovým mikrokontrolérem. Výběr zdůvodněte.
4. Návrh meteostanice rozpracujte. Zvolte vhodné snímače environmentálních veličin. Berte v úvahu možnost rozšíření od další funkce. Navrhněte zapojení elektronického obvodu, strukturu databáze, způsob komunikace meteostanice s databázovým systémem a způsob online vizualizace pomocí webové aplikace.
5. Návrh zrealizujte. Naprogramujte firmware meteostanice v jazyku C/C++. Naprogramujte databázový systém, webovou aplikaci pro vizualizaci.
6. Ověřte funkčnost a správnost vašeho řešení.

Rozsah grafických prací:	dle potřeby
Rozsah pracovní zprávy:	cca 45 stran + přílohy
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

[1] ĎAĎO, Stanislav a Marcel KREIDL. *Senzory a měřicí obvody*. Praha: ČVUT Praha, 1996. ISBN:80-01-01500-9 (K dispozici u vedoucího DP)
[2] FRAIN, Ben. *Responsive Web Design with HTML5 and CSS*, 3th. edition. Birmingham,UK:Pakt Publishing, 2022. ISBN 978-1803242712. (K dispozici u vedoucího DP)
[3] CHERNY, Boris. *Programming TypeScript*. Springfield, CA-US: O'Reilly Media, 2019. ISBN 978-1-492-03765-1.
[4] KLOBOUČEK, Jan. *Snímače v průmyslu (skriptum)*. Liberec: Technická univerzita v Liberci, 2012. ISBN 978-80-7372-828-1. (K dispozici u vedoucího DP)
[5] MASSE, Mark. *Rest API Design Rulebook*. CA-US: O'Reilly Media, 2011. ISBN 978-1449310509. (K dispozici u vedoucího DP)
[6] NORRIS, Donald. *Programming with STM32 – Getting Started with the Nucleo Board and C/C++*. NY-US: McGraw-Hill Education, 2018. ISBN 1260031314. (K dispozici u vedoucího DP)
[7] ONER, Vedat Odat. *Development IoT Projects with ESP32*. Birmingham, UK: Pakt Publishing, 2001. ISBN 978-1-83864-116-0. (K dispozici u vedoucího DP)
[8] PETRICK, Bill. *Learn harmony v3 for PIC32MZ*. WA-US: KDP Publishing, 2021. ISBN 979-1-985531277. (K dispozici u vedoucího DP)
[9] PRATA, Stephen. *Mistrovství v C++ – Kompletní průvodce vývojáře*, 4. aktualizované vydání. Praha: Computer Press, 2016. ISBN 978-80-251-3828-1. (K dispozici u vedoucího DP)
[10] GROFF, James a Paul WEINBERG. SQL Kompletní průvodce. Praha: Computer Press, 2005. ISBN 978-8-02510-369-2. (K dispozici u vedoucího DP)

Vedoucí práce:	Ing. Michal Moučka, Ph.D. Katedra výrobních systémů a automatizace
Datum zadání práce:	6. listopadu 2023
Předpokládaný termín odevzdání:	6. května 2025

L.S.

doc. Ing. Jaromír Moravec, Ph.D.
děkan

Ing. Petr Zelený, Ph.D.
vedoucí katedry

V Liberci dne 6. listopadu 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.

13. listopadu 2023 Bc. Radek Tvrzník

Meteorologická stanice se záznamem dat

Anotace

Diplomová práce se zabývá vývojem meteostanice se záznamem dat a online vizualizací časových průběhů environmentálních veličin. Teoretická část se zaměřuje na existující aplikace pro meteostanice, běhová prostředí JavaScriptu, frontendové a backendové frameworky, databázové systémy, mikrokontroléry a senzory. V praktické části je popsáno sestavení hardwarové části meteostanice, naprogramování mikrokontroléru, implementace serveru s databází a reverzní proxy, kontejnerizace, získávání předpovědi, systém notifikací, zálohování databáze, monitoring serveru, tvorba webové aplikace pro vizualizaci dat a nasazení webové aplikace i serveru.

Klíčová slova: ESP32, JavaScript, Docker, API, notifikace, nasazení, PWA, Svelte

Weather station with data recording

Annotation

The diploma thesis deals with the development of a weather station with data recording and online visualization of weather variables. The theoretical part focuses on existing weather station applications, JavaScript runtime environments, frontend and backend frameworks, database systems, microcontrollers and sensors. The practical part describes building the hardware part of the weather station, programming the microcontroller, implementing a server with database and reverse proxy, containerization, fetching forecasts, notification system, database backup, server monitoring, creating a web application for data visualization, and deploying the web application and server.

Keywords: ESP32, JavaScript, Docker, API, notifications, deployment, PWA, Svelte

Poděkování

Rád bych poděkoval vedoucímu mé diplomové práce Ing. Michalovi Moučkovi, Ph.D. za ochotu a odborné vedení při zpracování diplomové práce a též za veškerý mně věnovaný čas. Dále bych chtěl poděkovat své rodině za umožnění studia na vysoké škole, jejich neustálou podporu a ohleduplnost. Zároveň bych chtěl poděkovat Jakubovi Cieslarovi za povzbuzení a motivaci, abych se dále věnoval programování.

Obsah

Seznam použitých symbolů a zkratek

ACID	Atomicity, Consistency, Isolation a Durability
API	Application Programming Interface
CI/CD	Continuous integration / Continuous delivery
CRUD	Create, Read, Update a Delete
DevOps	Development operations
DOM	Document Object Model
DX	Developer experience
ECMA	European Computer Manufacturers Association
GC	Garbage collection
GHCR	GitHub Container Registry
HTTP	Hypertext Transfer Protocol
IDE	Integrated development environment
I/O	Input / Output
JSON	JavaScript Object Notation
MPA	Multi-page application
NoSQL	not only SQL
ORM	Object-relational mapping
SEO	Search engine optimization
SDK	Software development kit
SPA	Single-page application
SQL	Structured Query Language
SSE	Server-sent events
SSG	Static-site generation
SSR	Server-side rendering
TC39	Technical Committee 39
VAPID	Voluntary Application Server Identification
VPS	Virtual private server

1 Úvod

Na začátku roku 2022 jsem neměl prakticky žádné zkušenosti s programováním. I přesto jsem zasvětil velkou část svého volného času učení se programování. Jedním z řady projektů, které jsem během mého samostudia vytvořil, byla meteostanice, která je předmětem právě této diplomové práce.

Tato diplomová práce popisuje tvorbu meteostanice, která se skládá ze tří hlavních částí: Hardwarová část zabývající se záznamem dat, klientská část odpovědná za zobrazování všech dat uživatelům a serverová část sloužící jako prostředník mezi hardwarovou částí, klientem, databází a všemi dalšími službami. V této práci jsou kombinovány znalosti z několika oborů včetně elektroniky, programování mikrokontrolérů, vývoje fullstack aplikací, devops, a dalších.

2 Teoretická část

Teoretická část této diplomové práce se zabývá základními poznatky a technologiemi relevantní pro její vypracování. V první části jsou popsány existující klientské aplikace pro meteostanice, současné možnosti pro tvorbu aplikací a některé technologie, které se k jejich tvorbě používají. Další část je věnována popisu technologií použitých na serveru a databázovým systémům. V poslední části jsou probrány hardwarové součásti meteostanice jako jsou mikrokontroléry, vývojové desky a senzory meteorologických veličin.

2.1 Klient

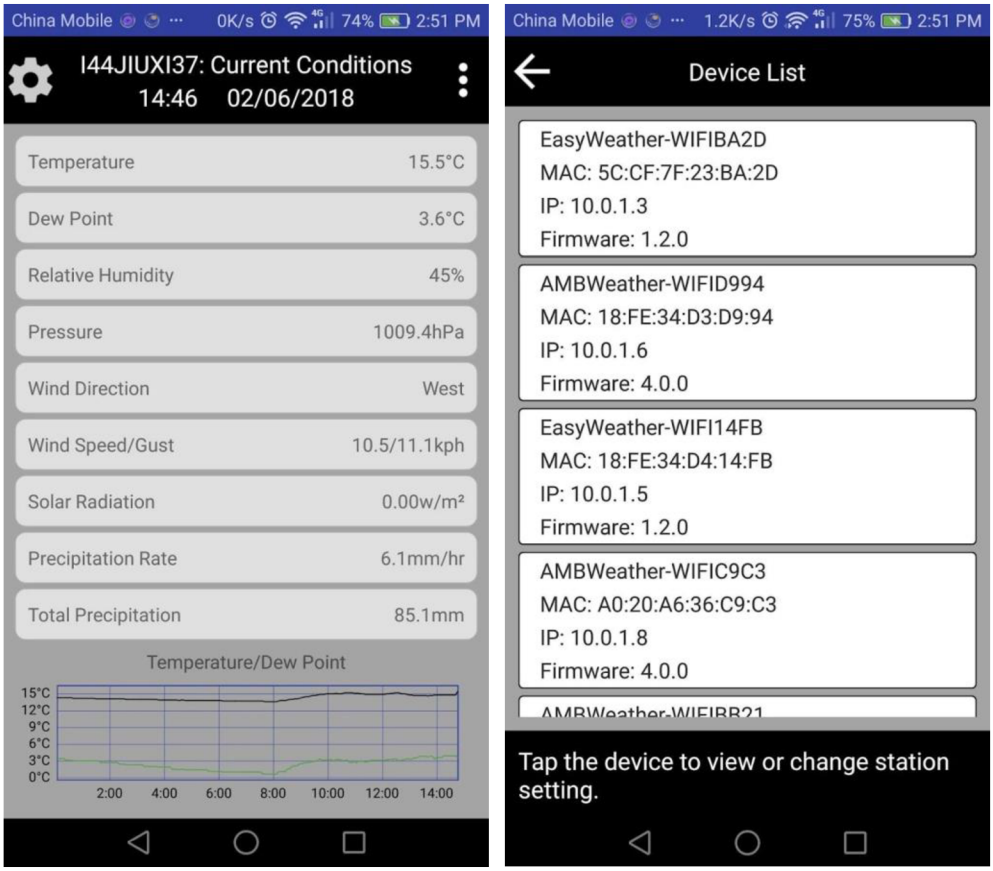
Součástí zadání je vytvořit webovou aplikaci pro vizualizaci dat z meteostanice. V této kapitole jsou blíže popsány některé existující aplikace pro meteostanice, současný stav platforem pro tvorbu aplikací a některé z nejpoužívanějších frameworků pro tvorbu moderních webových aplikací.

2.1.1 Existující aplikace pro meteostanice

Hlavním požadavkem při hledání existujících řešení byla možnost dlouhodobého záznamu a uchovávání naměřených dat. Tento požadavek mnoho aplikací pro meteostanice nesplňuje, nicméně jsou zde uvedeny některé aplikace, které umožňují alespoň omezené ukládání dat po delší dobu a přístup k historickým datům.

2.1.1.1 WS View

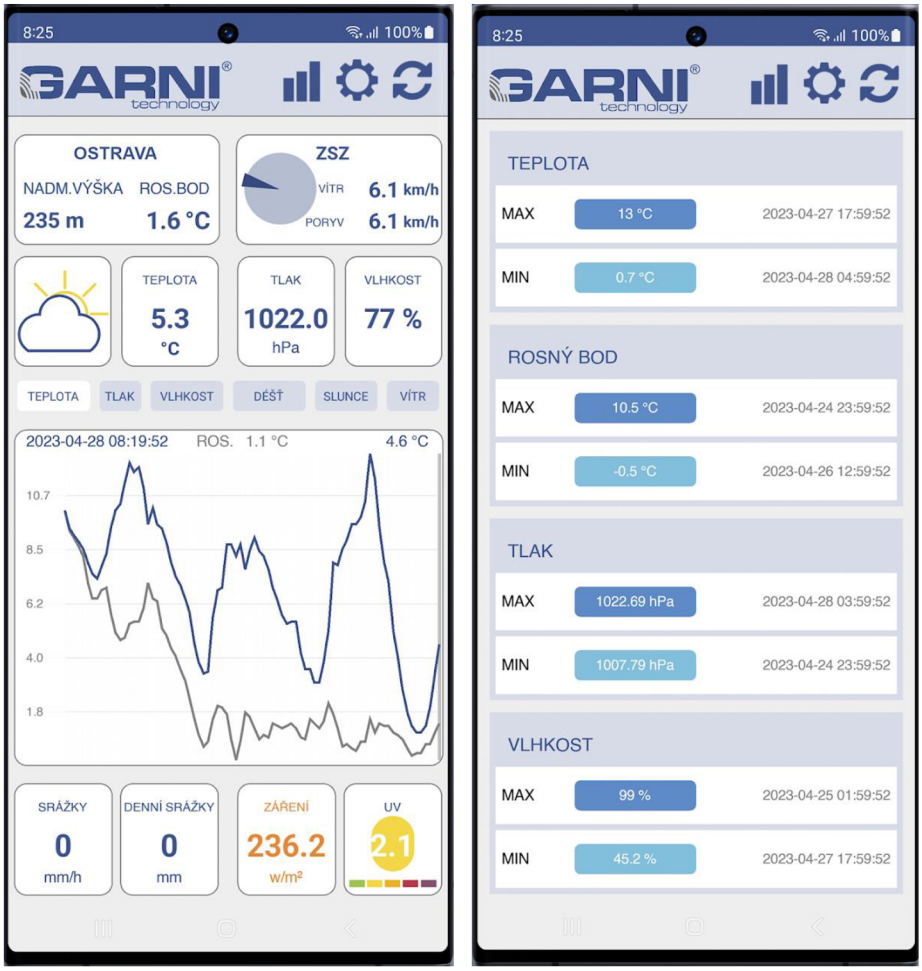
Aplikace WS View je využívána např. meteostanicemi od společnosti Gogen a umožňuje spárování s meteostanicemi i jiných značek. Aplikace obsahuje také dashboard pro sledování aktuálních hodnot. Podle uživatelských hodnocení je ale tato aplikace velmi nepřehledná a nespolehlivá. Možnosti pro sledování historických dat jsou velmi omezené. Aplikace má pouze anglickou lokalizaci [1].



Obr. 1 Uživatelské rozhraní aplikace WS View [1]

2.1.1.2 Garni

Aplikace Garni je určena pro meteostanice stejnojmenného českého výrobce. Využívá služby Weather Underground, ze které čerpá data. Aplikace má moderní uživatelské rozhraní a nabízí několikadenní historická data. Velkou nevýhodou je, že v základním balení meteostanice obsahuje pouze jeden aktivační kód a aplikaci lze tak používat pouze na jednom zařízení. Pro více aplikací je nutné kontaktovat výrobce [2, 3].



Obr. 2 Uživatelské rozhraní aplikace Garni [2]

2.1.1.3 WeatherLink

Americký výrobce profesionálních meteostanic Davis má také vlastní aplikaci WeatherLink. Data mohou být posílána přímo do konzole meteostanice nebo na cloudové úložiště Davisu. Základní verze aplikace umožňuje prohlížení aktuálních dat a shrnutí historických dat za poslední den / měsíc / rok. Pro podrobnější prohlížení historických dat je nutné zakoupit vyšší verzi stanice nebo profesionální verzi aplikace. Společnost Davis vyrábí mnoho různých modelů meteostanic pro profesionální použití primárně v zemědělství. Ceny nejlevnějších meteostanic začínají na 13 000 Kč [4, 5].

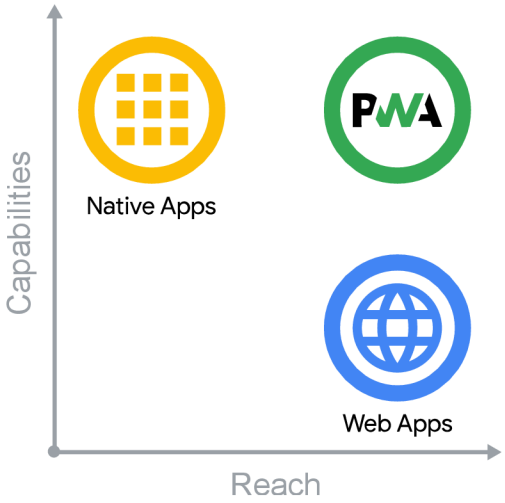
Obr. 3 Uživatelské rozhraní aplikace WeatherLink [4]

2.1.2 Webové aplikace

V dnešní době existují 3 hlavní platformy, pro které lze vytvořit aplikaci: desktopové (počítačové) aplikace, mobilní aplikace a webové aplikace.

Desktopové a mobilní aplikace jsou funkčně velmi obsáhlé a spolehlivé. Jsou všudypřítomné na domovských obrazovkách, docích a hlavních panelech. Většinou fungují bez ohledu na internetové připojení. Spouštějí se ve vlastním nezávislém prostředí. Mohou číst a zapisovat do souborů, přistupovat k hardwaru připojeném přes USB, Bluetooth nebo jiný port a dokonce mohou komunikovat s daty uloženými v zařízení jako jsou kontakty a události v kalendáři. Aplikace pro konkrétní platformu působí jako součást zařízení, na kterém jsou spuštěny. Hlavní nevýhodou desktopových a mobilních aplikací je, že fungují

pouze na jedné platformě. Pokud chceme, aby aplikace byla přístupná z obou platforem, tak je potřeba vytvořit a udržovat 2 samostatné aplikace. Tento problém dále komplikují jednotlivé operační systémy na dané platformě – Windows, macOS, Linux na počítačích a na mobilních zařízení Android a iOS [6].



Obr. 4 Schopnosti vs dosah nativních, webových a PWA aplikací [6]

Webové aplikace na druhou stanu může používat kdokoliv, kdekoliv a na jakémkoliv zařízení a operačním systému s jedinou kódovou základnou. V kombinaci s přirozenou provázaností webu umožňují webové aplikace jednoduše sdílet informace s kýmkoliv a kdekoliv. Lze například vytvořit lokální aplikaci pro videochat promocí WebRTC, geolokace a push notfikací [6].

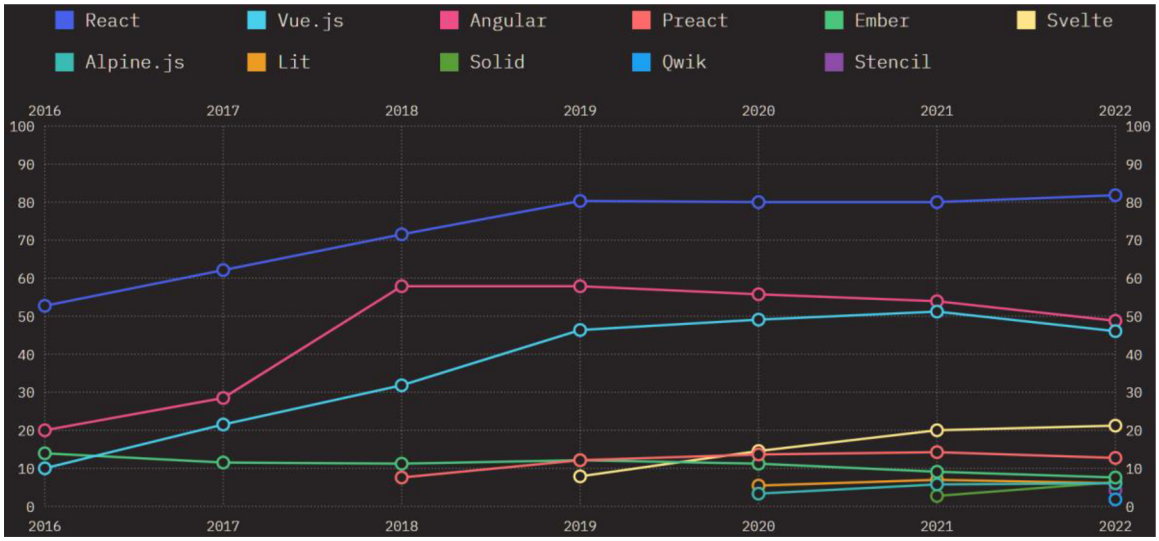
S příchodem WebAssembly mohou vývojáři používat i jiné programovací jazyky, jako jsou třeba C, C++ a Rust, a přenést na web tak desítky let zkušeností a rozvinutých ekosystémů. Například aplikace Photoshop a Figma lze díky WebAssembly plně používat z webového prohlížeče [7].

Až donedávna mohly tyto pokročilé funkce využívat pouze nativní aplikace. Některé možnosti jsou sice stále mimo dosah webu, ale nová a připravovaná rozhraní API se toto snaží změnit a rozšiřují možnosti webu o funkce, jako je přístup k souborovému systému, notifikacím, USB, ovládání přehrávaných médií, přístup ke kontaktům, Bluetooth a mnoho dalších API [8].

Nainstalované progresivní webové aplikace (PWA) se spouštějí v samostatném okně místo v kartě prohlížeče. Lze je spustit z domovské obrazovky mobilních zařízení a plochy počítačů podobně jako nativní aplikace. Mohou být nastaveny jako výchozí aplikace pro zpracování různých typů souborů. Běžný uživatel pak po instalaci vůbec nepozná rozdíl mezi PWA a klasickou nativní aplikací pro svůj operační systém. Dále díky service workerům a ukládání do mezipaměti mohou PWA plně fungovat i bez připojení k internetu [6].

2.1.3 Frontend frameworky

Web je dnes nezbytnou součástí moderního života a 98 % webových stránek používá JavaScript. Uživatelé na webu píší dokumenty, spravují své finance, poslouchají hudbu, sledují filmy a komunikují s ostatními pomocí textu, zvuku nebo video chatu. Web nám umožňuje dělat věci, které byly dříve možné pouze v nativních aplikacích nainstalovaných v počítači. Moderní JavaScriptové frameworky výrazně usnadňují vytváření vysoce dynamických a interaktivních aplikací. Framework je knihovna, která nabízí určité přístupy, jak vytvářet software. Tyto přístupy zajišťují předvídatelnost a jednotnost celé aplikace. Díky frameworkům lze aplikace škálovat a je snazší je udržovat po dlouhou dobu životnosti softwaru. JavaScriptové frameworky pohánějí většinu webových aplikací na dnešním moderním webu. V následujících podkapitolách jsou popsané některé z nejpoužívanějších JavaScriptových frontendových frameworků [9].



Obr. 5 Výsledky průzkumu nejpoužívanějších frontendových frameworků [10]

2.1.3.1 React

Zdaleka nejpoužívanější frontend framework je React. React vznikl ve společnosti Facebook, kde byl již interně využívaný několik let a v roce 2013 byl vydaný jako bezplatný open-source projekt. React je oficiálně klasifikovaný jako knihovna, nicméně mnoho vývojářů ho považuje za plnohodnotný framework. Jeho hlavní funkcí je vykreslování komponent uživatelského rozhraní, k čemuž využívá virtuální objektový model dokumentu (DOM) [9, 11].

DOM je strukturovaná reprezentace aktuálního stavu HTML dokumentu, ze které webový prohlížeč vykresluje dané webové stránky. React v paměti uchovává virtuální kopii skutečného DOM, kterou aktualizuje tak, aby vždy odpovídala současnému stavu dat. Poté React porovnává virtuální DOM se skutečným, aby zjistil, co přesně se změnilo – diffing (porovnávání). Následně nalezne nejoptimálnější způsob, jak aktualizovat části skutečného DOM, aniž by došlo k znovunačtení celé stránky – rekonciliace. Hlavní nevýhodou tohoto přístupu je, že čím složitější jsou webové stránky, tím pomalejší je proces porovnávání virtuálního a skutečného DOM a tím pádem i samotná aktualizace stránky je pomalejší. Novější frameworky jako třeba Svelte nebo Solid virtuální DOM nemají vůbec a využívají efektivnější přístupy pro aktualizaci skutečného DOM [12, 13].

React se velmi často používá pro tvorbu jednostránkových aplikací (SPA), které načtou jedinou HTML stránku a dále ji už jenom pomocí JavaScriptu dynamicky aktualizují podle toho, jak uživatelé s aplikací interagují. SPA aplikace vytvářejí plynulý a nepřerušovaný dojem podobný nativním desktopovým nebo mobilním aplikacím. React se také používá pro tvorbu mobilních aplikací pomocí knihovny React Native. Lze tak vytvářet webové i mobilní aplikace s relativně podobnou syntaxí bez nutnosti učit se zcela nový programovací jazyk [14].

Ačkoliv byl React původně navržen pro použití v prohlížeči, lze ho provozovat na serveru s Node.js a vytvářet plně SSR aplikace např. pomocí meta-frameworku Next.js. Tato možnost byla již delší dobu velmi populární a v roce 2023 proto byly do Reactu přidány oficiální serverové komponenty umožňující načítání a vykreslování dat výhradně na straně serveru. Výsledné vygenerované HTML se pak streamuje přímo do prohlížeče a na straně klienta se hydratuje a podle potřeby prokládá s dalšími klientskými komponentami [15, 16].

Oproti jiným frameworkům je React velmi flexibilní a vývojářům většinou nevnucuje žádné konkrétní řešení. Například Angular používá pro správu stavu a pro routing vlastní řešení, podobně jako Vue nebo Svelte. React v těchto oblastech žádné oficiální řešení nenabízí a většinou jsou tak používány knihovny třetích stran jako je např. React Router pro routing a Redux pro správu stavu [11, 14].

2.1.3.2 Vue

Vue.js, často označovaný jednoduše jako Vue, je open-source frontend framework pro vytváření interaktivních webových uživatelských rozhraní. Podobně jako React a jiné frameworky i Vue používá deklarativní vykreslovací systém. Tato funkce umožňuje vývojářům specifikovat pouze jak má uživatelské rozhraní vypadat na základě aktuálního stavu dat. Vývojáři se ale už nemusí zabývat tím, jak dosáhnout tohoto požadovaného stavu, což je práce frameworku samotného, který automaticky aktualizuje zobrazení bez dodatečných instrukcí. Pro aktualizace částí uživatelského rozhraní Vue používá virtuální objektový model dokumentu (DOM) jako React. Vue v základu poskytuje animace a přechodové efekty pro vkládání nebo odstraňování prvků z DOM. Architektura Vue je stejně jako v Reactu založená na komponentách, které podporují opakované použití kódu a modularitu aplikací. Pokud je aplikace citlivá na optimalizaci pro internetové vyhledávače (SEO), Vue poskytuje prvořadé API pro vykreslení aplikace na straně serveru (SSR). To umožňuje serveru okamžitě odeslat již vykreslený HTML dokument, takže koncoví uživatelé mohou obsah vidět okamžitě, zatímco na straně prohlížeče se načítá JavaScript. Vue pak aplikaci v prohlížeči hydratuje, aby stránka byla interaktivní. Tohoto využívají meta-frameworky založené na Vue jako je např. Nuxt [17, 18].

Jedním z hlavních aspektů Vue je jeho filozofie minimálních nároků a možnost postupné integrace do aplikace. Podobně jako jQuery lze i Vue používat jako samostatný skript soubor. Je to nejjednodušší způsob jako integrovat Vue s již existujícím backendem renderující většinu HTML, anebo pokud frontendová logika není dostatečně složitá, aby byl build krok nutný. Pomocí Vue lze také vytvářet standardní webové komponenty, které lze použít kdekoliv. Je tak možné integrovat Vue do již existujících aplikací nebo dokonce do aplikací vytvořených pomocí jiných frameworků [18, 19].

2.1.3.3 Svelte

Svelte je komponentový framework podobně jako React nebo Vue, ale s důležitým rozdílem. Tradiční frameworky umožňují psát deklarativní stavem řízený kód za cenu toho, že prohlížeč musí provést dodatečnou práci, aby tyto deklarativní struktury převedl do skutečného stavu DOM. Převod pomocí technik jako je virtuální DOM může být náročný na procesor a paměť. Namísto toho Svelte v moment kompilace aplikace převede všechny komponenty na vysoce optimalizovaný imperativní JavaScript kód, který precizně aktualizuje skutečný DOM. Výsledkem tohoto přístupu je lepší výkon aplikací a menší velikosti JavaScriptových balíčků, které musí uživatel stahovat. Na rozdíl od Reactu nebo Vue, nemusí Svelte v prohlížeči načítat celou celou knihovnu frameworku, což má pozitivní dopad na rychlost aplikace. Aplikace vytvořené pomocí Svelte obecně běží rychleji na zařízeních s nižším výkonem jako jsou např. mobilní telefony nebo méně výkonné počítače a načítání webových stránek přes slabší internetové spojení je také podstatně svižnější [20, 21].

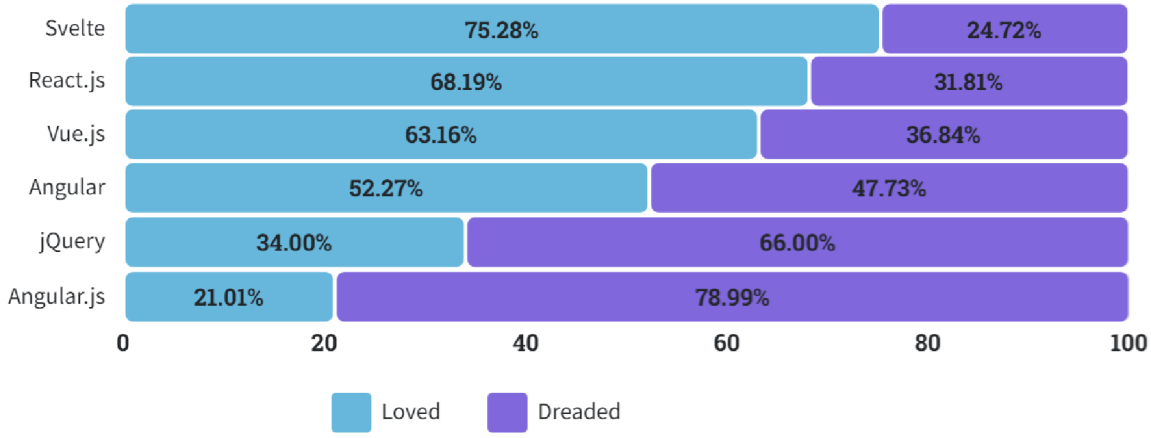
Svelte se úzce drží klasického modelu vývoje webových aplikací pomocí HTML, CSS a JavaScriptu. Oproti jiným frameworkům Svelte tyto základní jazyky pouze lehce rozšiřuje o vlastní syntaxi a nijak je zásadně nemodifikuje. Výsledkem je framework, který je snadno pochopitelný i pro vývojáře bez předchozích zkušenost. Svelte doplňuje CSS o funkci scopingu, která umožňuje definovat styly komponenty uvnitř komponenty samotné. Styly pak neovlivňují ostatní komponenty a nehrozí kolize CSS. Toto řešení je mnohem elegantnější oproti Reactu, kde zapouzdření CSS je mnohem složitější a často se používají neoficiální řešení třetích stran. Další velkou výhodou Svelte je systém zprávy stavu a reaktivita. Správa stavu v Reactu je oproti Svelte složitější a pro komplexnější aplikace se často používají komplikované nástroje jako je Redux. Svelte má pro správu stavu zabudované velmi robustní řešení, které je v základu velmi jednoduché. Ve výchozím nastavení jsou proměnné reaktivní a není potřeba psát žádný speciální kód. Pro složitější případy pak lze využít možností které nabízí Svelte samotné. Reaktivita by měla být ještě dále vylepšena ve Svelte 5, které má být vydáno v roce 2024 a jehož beta verze má velice kladné ohlasy [20, 22, 23].

Podobně jako jiné frameworky má i Svelte vlastní meta-framework. SvelteKit je ale na rozdíl od ostatních meta-frameworků vyvíjený stejnými vývojáři jako Svelte samotné. SvelteKit je velmi úzce propojený se Svelte a jeho největší výhodou je kompatibilita s většinou moderních architektur webových stránek včetně

jednostránkových aplikací (SPA), vícestránkových aplikací (MPA), serverově vykreslovaných aplikací (SSR) a staticky generovaných stránek (SSG). SvelteKit nabízí vše od základních očekávaných funkcí, jako je např. router až po pokročilejší funkce. Jeho rozsáhlý seznam funkcí zahrnuje optimalizovanou kompilaci pro načítání minimálního množství kódu, podpora offline stránek a progresivních webových aplikací (PWA) pro offline použití, automatické přednačítání stránek, nastavitelné vykreslování částí aplikace na serveru a jiných částí v prohlížeči a mnoho dalších. Hlavním rozdílem mezi SvelteKitem a ostatními meta-frameworky je, že například v ekosystému Reactu se React používá primárně na SPA aplikace a pokud chceme vytvořit SSR aplikaci, tak musíme sáhnout po dalším frameworku jako je např. Next.js. Podobně tomu je i v případě Vue (primárně SPA aplikace) a Nuxt (SSR a další). V ekosystému Svelte se ale SvelteKit běžně používá na všechny druhy aplikací a typ architektury lze snadno zvolit pomocí oficiálních adaptérů. I když je možné použít Svelte bez SvelteKitu, oficiální dokumentace a tutoriály směřují vývojáře na používání SvelteKitu, který Svelte obohacuje o mnoho funkcí, usnadňuje vývoj nových aplikací a stejně jako Svelte je velmi přívětivý pro nové vývojáře [24, 25].

Hlavní výhodou a zároveň nevýhodou Svelte je, že se jedná o poměrně nový framework. První verze Svelte byla vydaná v roce 2016 a SvelteKit byl vydaný v roce 2022. Ekosystém Svelte je proto menší z hlediska počtu nástrojů, podpory a pracovních příležitostí než u jiných starších frameworků. I když počet existujících Svelte knihoven je menší než u ostatních frameworků, je nutné si uvědomit, že používání čistě JavaScriptových knihoven např. v Reactu může být poměrně komplikované. Obzvláště nepraktické pak mohou být knihovny, které jakkoliv manipulují s DOM. Proto v ekosystému Reactu existuje mnoho podpůrných knihoven, které pouze obalují JavaScriptové knihovny v React kódu, aby je vůbec bylo možné použít v Reactu. Protože je ale Svelte velmi podobné čistému HTML, CSS a JavaScript kódu, tyto podpůrné knihovny většinou nejsou potřeba a lze jednoduše použít samotné JavaScriptové knihovny bez zbytečného kódu a závislostí navíc.

Na závěr je nutné dodat, že ačkoliv komunita Svelte je zatím relativně malá oproti dlouho zavedeným frameworkům jako React, tak se Svelte v posledních letech stabilně umisťuje na prvních místech vývojářských průzkumů jako nejoblíbenější frontend framework – viz. obr. 6 [10, 26, 27]. Svelte začíná také používat mnoho velkých společností mezi které patří např. Apple, Spotify, Ikea, Decathlon, Cloudflare a další [28].



Obr. 6 Výsledky průzkumu oblíbenosti frontendových frameworků [26]

2.1.3.4 Shrnutí frontend frameworků

React je bezesporu nejpoužívanější framework s největší komunitou vývojářů a knihoven. Z hlediska pracovních příležitostí v tvorbě frontendu webových aplikací je React stále jedničkou. Navzdory své popularitě však React často zaostává, pokud jde o DX (vývojářská přívětivost) a mnoho na první pohled menších nepříjemností, jako je např. nutnost psát spoustu kostrbatého a často opakujícího se kódu, který jednoduše při použití alternativních frameworků není potřeba vůbec psát. Naproti tomu Svelte vyniká v oblasti DX, díky své modernější a intuitivnější syntaxi. Z hlediska výkonu Svelte také dominuje díky svému kompilátoru, který zajišťuje mnohem menší velikosti webových stránek a menší zátěž na zařízeních se slabším výkonem a připojením. Výkon a DX budou dále ještě podstatně zlepšeny v nové verzi Svelte 5, která by měla být vydána roce 2024. V neposlední řadě na rozdíl od Reactu, kde se vývojáři většinou musí uchýlit ke skládání několika různých knihoven dohromady, nabízí SvelteKit integrovaný a ucelený framework, který pokrývá všechny podstatné aspekty moderního vývoje webových aplikací. Je však nutné mít na paměti, že všechny tyto výhody jsou možné hlavně díky novosti Svelte, což je dvousečná zbraň. Ekosystém knihoven Svelte není tak rozsáhlý jako ekosystém Reactu. Vue a do jisté míry i Angular se nachází někde mezi Reactem a Svelte, pokud jde o rozšířenost, DX a pracovní příležitosti. Na základě předchozích zkušeností se Svelte i Reactem byl pro tuto práci vybrán Svelte společně se SvelteKitem.

2.2 Server

Tato kapitola je věnována hlavním technologiím běžícím na serveru, který slouží jako prostředník mezi meteostanicí, databází a klientem. Jako hlavní jazyk pro server samotný byl vybraný JavaScript (TypeScript) kvůli předchozím zkušenostem s jazykem a masivní open-source komunitě. Jsou zde popsána nejpopulárnější běhová prostředí pro JavaScript a jejich současný stav. Dále jsou zde rozebrány nejpoužívanější serverové frameworky pro JavaScript a některé přední open-source databázové systémy. Závěr této kapitoly je věnovaný popisu kontejnerů a významu kontejnerizace aplikací v současné době.

2.2.1 JavaScript na serveru

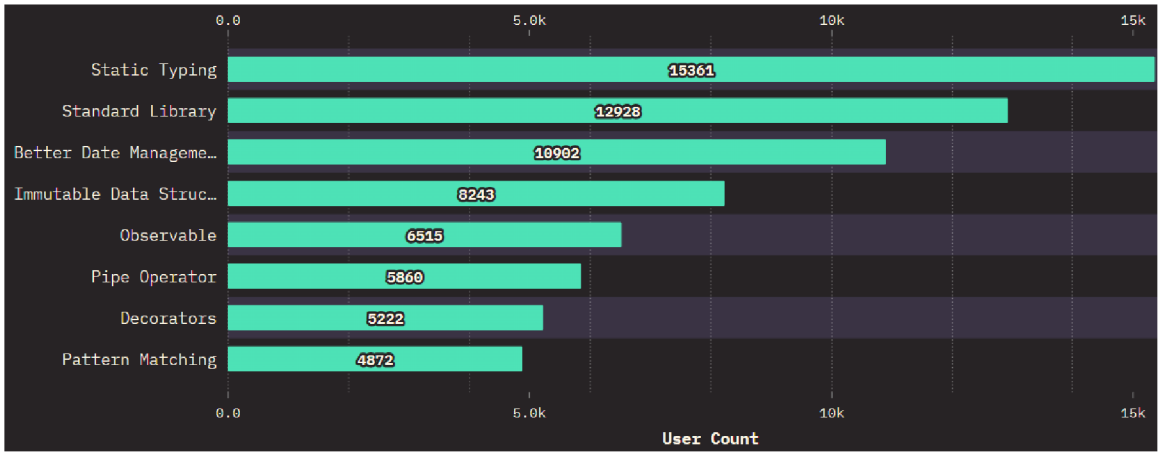
JavaScript je interpretovaný (just-in-time kompilovaný) programovací jazyk. Ačkoli je nejznámější jako skriptovací jazyk pro webové stránky, je používaný v mnoho jiných prostředí než jen v prohlížeči. JavaScript podporuje objektově orientované, funkcionální i procedurální programování [29]. Ačkoli se Java a JavaScript podobají názvem a syntaxí, oba jazyky jsou odlišné a značně se liší svojí funkcionalitou a návrhem.

JavaScript byl vytvořen v roce 1995 Brendanem Eichem ve společnosti Netscape. O dva roky později byla vydána první specifikace jazyka ECMAScript sloužící jako společný bod pro standardní specifikaci JavaScriptu, kterou by mohli dodržovat všichni tvůrci prohlížečů [2, 7]. Návrh specifikace ECMAScriptu je v současné době udržován otevřeně na GitHubu [30] a každoročně je vytvářena nová verze specifikace nesoucí jméno daného roku (např. *ES2023*) [31, 32]. Případné revize jazyka jsou schvalovány prostřednictvím komplexního čtyřstupňového procesu návrhů [33, 34] skrz komisi, která se schází 6x ročně a jejíž členové jsou např. společnosti jako Apple, Facebook, Google, IBM, Microsoft a další [35].

JavaScript je dynamicky typovaný, což znamená, že některé typy jsou přiřazeny implicitně na základě prováděné operace. Kupříkladu jedné proměnné může být přiřazena celočíselná hodnota, později řetězec znaků, pole, třída, či jiný datový typ. Při převodu proměnné z jednoho typu na druhý v JavaScriptu však existuje mnoho zvláštností, které jsou často terčem kritiky [36, 37].

V uplynulém desetiletí se poměrně úspěšně osvědčilo statické ověřování typů. Microsoft, Google a Facebook vydali TypeScript, Closure Compiler a Flow. Tyto projekty

byly velkými investicemi do JavaScriptu s cílem zvýšení produktivity vývojářů, které bylo zaznamenáno u staticky typovaných jazyků. V případě TypeScriptu, Flow a dalších přinesly tyto varianty jazyka pohodlnou syntaxi pro deklarování a používání typů v JavaScriptu. Tato syntaxe většinou neovlivňuje samotný běh programu a v praxi se převod těchto variant jazyka na obyčejný JavaScript omezuje na pouhé vymazání typů [38].



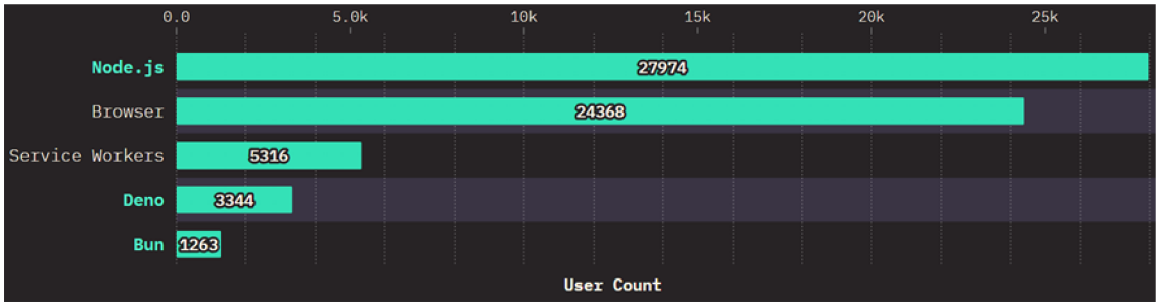
Obr. 7 Funkce, které podle vývojářů chybí v JavaScriptu nejvíce [10]

Silná poptávka vývojářů po ergonomické syntaxi statického typování vedla ke vzniku ECMAScript návrhu, jehož cílem je umožnit vývojářům vkládat do kódu typové anotace a následně kontrolovat tyto anotace pomocí externích nástrojů před spuštěním kódu, podobně jako je tomu u staticky typovaných jazyků. JavaScriptový engine by typy ignoroval a považoval je za komentáře. Tento návrh by umožňoval vývojářům psát a spouštět programy s typy bez nutnosti transpilace kódu [38]. Dle poznámek z poslední schůze TC39 v září 2023 vyplývá, že na návrhu se stále intenzivně pracuje a návrh zatím zůstává ve fázi 1 [39]. Prvotní návrh tedy zatím není plně dokončen a není zcela jisté, zda návrh bude implementovaný do standardu ECMAScript a posléze do JavaScriptu.

Nejpopulárnější variantou JavaScriptu s typy je aktuálně TypeScript od Microsoftu. Použití TypeScriptu je bezplatné a zdrojový kód je veřejně dostupný pod licencí Apache. TypeScript přidává statické typování s volitelnými typovými anotacemi, avšak nejedná se o „pravé“ statické typování, které mají přísnější jazyky jako Java, nebo Rust. TypeScript je navržen jako nadstavba pro JavaScript a klade důraz na kompatibilitu s rozsáhlým

stávajícím ekosystémem JavaScriptových knihoven. Jakýkoli platný JavaScriptový kód je také platný TypeScript kód. Tento přístup ale může vést k případům, kdy TypeScript nemusí zachytit některé chyby, které by byly zachyceny v jazycích s přísnějším typovým systémem již při kompilaci, kdežto v JavaScriptu až při běhu daného kódu [40].

JavaScript je interpretovaný programovací jazyk a jeho kód potřebuje ke spuštění běhové prostředí (runtime). Při vývoji webových stránek je nejběžnější běhové prostředí to, které poskytuje daný webový prohlížeč. Každý prohlížeč má svůj vlastní JavaScriptový engine, který je zodpovědný za interpretaci a běh kódu. Například Google Chrome používá engine V8, Mozilla Firefox používá SpiderMonkey a Safari používá JavaScriptCore. Jak již bylo zmíněno, JavaScript se používá i mimo prohlížeče pro tvorbu webových serverů. I v tomto případě je nutné mít na serveru běhové prostředí odpovědné za běh JavaScriptové kódu na serveru [32].



Obr. 8 Výsledky průzkumu nejpoužívanějších JavaScriptových běhových prostředí [10]

V následujících podkapitolách jsou podrobněji popsána nejčastěji používaná běhová prostředí pro JavaScript mimo webový prohlížeč.

2.2.1.1 Node.js

Node.js je open-source běhové prostředí JavaScriptu napsané v jazyce C++, které je primárně určeno k vytváření škálovatelných webových aplikací. Node.js běží na enginu Chrome V8 a umožňuje spouštět JavaScriptový kód i mimo webové prohlížeče. Node.js vytvořil Ryan Dahl v roce 2009. Jádrem Node.js je tzv. smyčka událostí (event loop), která zpracovává veškeré požadavky uživatele. Pomalejší blokující operace, jako je např. čtení a zápis do souborů, nebo síťové požadavky na ostatní servery, běží na samostatných vláknech nezávisle na hlavním vlákně Node.js, které tak není blokováno a může běžet dále. Díky tomu je Node.js vhodné pro vývoj škálovatelných systémů [41].

Součástí Node.js je balíčkový systém *npm*, pomocí kterého lze instalovat a spravovat všechny potřebné knihovny a také sdílet a distribuovat JavaScriptový kód. Součástí *npm* je i online registr veřejně dostupných balíčků a knihoven, který v září 2022 obsahoval více než 2,1 miliónů balíčků [42] a v roce 2016 bylo každý týden odtud staženo přes 1 miliardu balíčků [43], což z *npm* činí největší úložiště kódu jednotného programovacího jazyka na světě [44].

2.2.1.2 Deno

Deno je open-source běhové prostředí pro JavaScript, TypeScript a WebAssemly běžící na enginu V8 podobně jako Node. Deno bylo oficiálně představeno v roce 2018 na konferenci JSConf EU samotným autorem běhového prostředí Node.js, Ryanem Dahlem, v jeho vystoupení s názvem „10 věcí, kterých lituji na Node.js“ [45]. Cílem tohoto nového běhového prostředí bylo opravit dlouhý seznam všech chyb a nedostatků Node.js, které přiznal samotný autor. Mezi tyto nedostatky patří například bezpečnostní oprávnění běhového prostředí, způsob spravování závislostí pomocí *package.json, node_modules* a mnoho dalších [46].

Bezpečnost je jednou z hlavních oblastí, které Dahl litoval na Node.js. Deno naproti tomu spouští veškerý kód v sandboxu, což znamená, že běhové prostředí v základu nemá žádný přístup k souborovému systému, síti, environmental variables, nemůže spouštět jiné skripty a další. Všechna oprávnění musí být programu explicitně povolena přes parametry zadávané při spouštění programu [47].

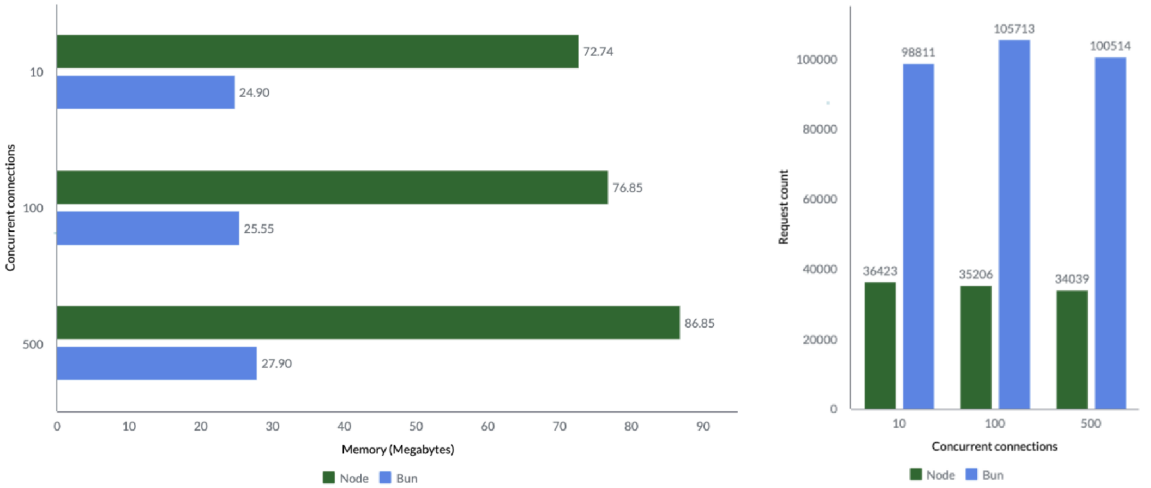
Deno podporuje načítání balíčků z URL, podobně jako JavaScript v prohlížeči. Díky tomu není Deno závislé na jediném centrálním registru jako je npm. Lze se tak také zbavit notoricky známých *node_modules* a *package.json*. Při spuštění programu Deno stáhne všechny použité balíčky a uloží je do mezipaměti pro budoucí použití [46].

Deno bylo původně napsáno v programovacím jazyce *Go*, avšak z důvodu zvýšení výkonu a odstranění dodatečného běhového prostředí a GC jazyka *Go* bylo Deno přepsáno do jazyka Rust [48].

2.2.1.3 Bun

Bun je rychlé open-source JavaScriptové běhové prostředí navržené jako drop-in náhrada za Node.js. Bun má za cíl nahradit spousty komplexních nástrojů, které jsou v dnešní době potřebné pro vývoj v JavaScriptu a integrovat je do jedné ucelené sady nástrojů pro vývojáře tvořící aplikace v JavaScriptu a TypeScriptu. Jeho součástí je správce balíčků, bundler, trasnpilátor kódu, spouštěč testů a další nástroje, které byly od základu napsané v programovacím jazyce Zig [49].

Hlavní předností Bunu je jeho rychlost. Na rozdíl od Node.js a Deno, které používají engine V8, běží Bun na enginu JavaScriptCore (JSC) využívaný v prohlížeči Safari. V8 a JSC mají odlišnou architekturu a optimalizační strategie. JSC dává přednost rychlejšímu spuštění a menší spotřebě paměti při mírně pomalejším běhu kódu. Naproti tomu V8 upřednostňuje rychlý běh kódu s větší optimalizací za běhu, což může vést k větší spotřebě paměti. Díky tomu je Bun rychlý a startuje až 4x rychleji než Node.js [50, 51]. Srovnávací testy od Ahmod [52], oficiální dokumentace [51, 53] a i dalších autorů tyto tvrzení potvrzují. Tyto testy jsou však často prováděny pomocí velmi jednoduchých testovacích programů na lokálním hardwaru a síti, což nereflektuje skutečné podmínky komplexních aplikací běžících na reálných webových serverech. Je tedy důležité přistupovat k těmto výsledkům opatrně a být si vědom jejich omezení.



Obr. 9 Porovnání spotřeby paměti a počtu úspěšných HTTP požadavků [52]

Mezi další výhody Bunu patří například zabudovaná podpora TypeScriptu, *.jsx* a *.tsx* souborů. Transpilátor Bunu před spuštěním vše překládá do samotného JavaScriptu. Cílem Bunu je také dosáhnout plné kompatibility s Node.js a všemi patřičnými API a moduly (process, Buffer, path, fs, http a další) [49].

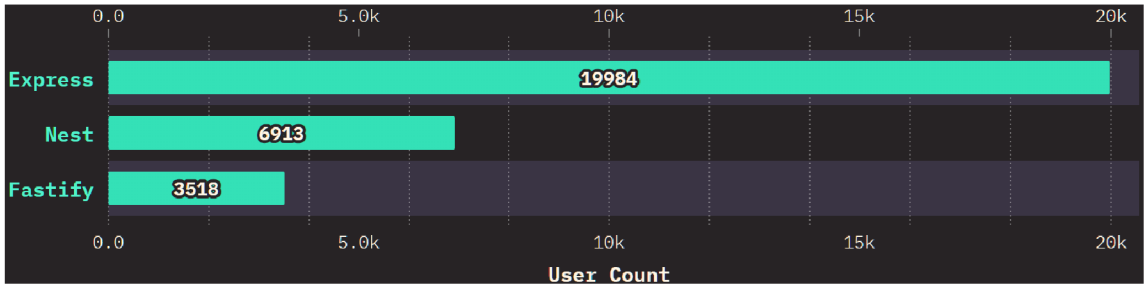
Ačkoliv se v současné době jeví Bun jako nejrychlejší JavaScriptové běhové prostředí, tak Bun s sebou nese jednu velkou nevýhodu a tou je stabilita. Vývoj Bunu začal v roce 2022 a 8.9. 2023 byla vydána verze 1.0 [51]. Hlavní prioritou Bunu je už dlouhou dobu zlepšení stability [54] avšak aktuálně je na oficiálním repozitáři Bunu nahlášeno více chyb typu *segmentation fault* a *core dumped,* než u současných konkurentů Node.js (vývoj od 2009) a Deno (vývoj od 2018) dohromady [55–57]. Tato nejistota ohledně stability může některé vývojáře od přechodu na toto běhové prostředí, ačkoliv jeho výkon je nesporný.

2.2.1.4 Shrnutí JavaScriptových běhových prostředí

Pro tuto práci byl vybrán Node.js jako běhové prostředí. Node je nejrozšířenější běhové prostředí s masivní komunitou vývojářů a je stále de facto standardem pro běh JavaScript na serveru, pro transpilaci JavaScriptu i na frontendu a pro všechny možné JavaScriptové nástroje. Node má bezesporu nejlepší kompatibilitu se všemi knihovnami a všemožnými nástroji, což ovšem nelze říci o Deno ani o Bunu. Bun má sice za cíl plnou kompatibilitu s Node.js a Deno přidalo podporu npm, nicméně plnou kompatibilitu s již existujícím ohromným ekosystémem JavaScriptových knihoven nemá ani jedno běhové prostředí. Bun stále trápí problémy se stabilitou, a i když je Bun jedno z nejrychlejších běhových prostředí, tak pro většinu serverových aplikací je rychlost Node.js dostačující a větší prioritu má stabilita. Node.js je tak stále nejvíce osvědčené běhové prostředí pro JavaScript.

2.2.2 JavaScriptové server frameworky

Pro vytvoření kompletního backendu v JavaScriptu je potřeba pouze běhové prostředí jako je například Node.js. S rostoucími požadavky projektu však roste i počet funkcí, které je potřeba implementovat. Většina projektů vyžaduje klíčové prvky jako je router, middleware, validace a další funkce. Řešení pro tyto komplexnější koncepty běhová prostředí většinou nenabízí a i kdybychom je ručně implementovali, výsledkem by pravděpodobně byl náš vlastní framework, kterému by nikdo jiný nerozuměl. Ve většině případů je tedy vhodné zvolit zavedený framework, který prošel mnoha iteracemi vývoje a byl použit ve velkém měřítku, což znamená, že mnoho chyb a bezpečnostních zranitelností již bylo objeveno a odstraněno.



Obr. 10 Výsledky průzkumu nejpoužívanějších JavaScriptových backend frameworků [10, 58]

2.2.2.1 Express

Express je nejpopulárnější Node.js backend framework a je základní knihovnou pro řadu dalších populárních webových frameworků jako je NestJS, Sails.js a další [58–60]. Express byl vyvinutý pro Node.js v roce 2010 a aktuálně je spravovaný společnosti IBM [61]. I když samotný Express je poměrně minimalistický framework, rozsáhlá komunita vývojářů vytvořila kompatibilní balíčky middlewaru, které řeší téměř jakýkoliv problém při vývoji webových serverů. Existují knihovny pro práci s cookies, relacemi, přihlašování uživatelů, URL parametry, validace POST dat, bezpečnostními hlavičkami a mnoho dalších [62].

2.2.2.2 Nest

Nest (NestJS) je framework pro vytváření webových aplikací na straně serveru Node.js. Nest plně podporuje a je vytvořený pomocí TypeScriptu, ale umožňuje použití i čistého JavaScriptu. Framework je silně zaměřený na objektově orientované programování a využívá funkce jako jsou dekorátory, dependency injection a další. Nest ve výchozím nastavení používán framework Express a volitelně jej lze nakonfigurovat pro použití s frameworkem Fastify. Hlavním cílem tohoto frameworku je vytvořit abstrakci nad těmito běžnými frameworky a poskytnout hotovou architekturu aplikací, která bude vždy podobná a nezávisle na projektu, ve kterém se Nest použije. Tato filozofie a architektura je silně inspirována frontendovým frameworkem Angular [58].

2.2.2.3 Fastify

Fastify je jeden z nejrychlejších backendových frameworků pro Node.js. V závislosti na složitosti kódu dokáže obsloužit až 30 tisíc požadavků za sekundu podle tvrzení vývojářů. Framework samotný je napsaný v JavaScriptu, ale jsou k dispozici oficiální soubory s typovými deklaracemi a je tak možné používat i TypeScript. Dalším cílem je poskytnout co nejlepší DX (vývojářská přívětivost) bez obětování výkonu a bezpečnosti [63]. Podobně jako Express, je i Fastify relativně minimalistický framework a jeho tým vývojářů také nabízí velmi rozsáhlou sbírku oficiálních pluginů a je možné použít i mnoho dalších pluginů vyvinutých komunitou [64].

2.2.2.4 Shrnutí JavaScriptových server frameworků

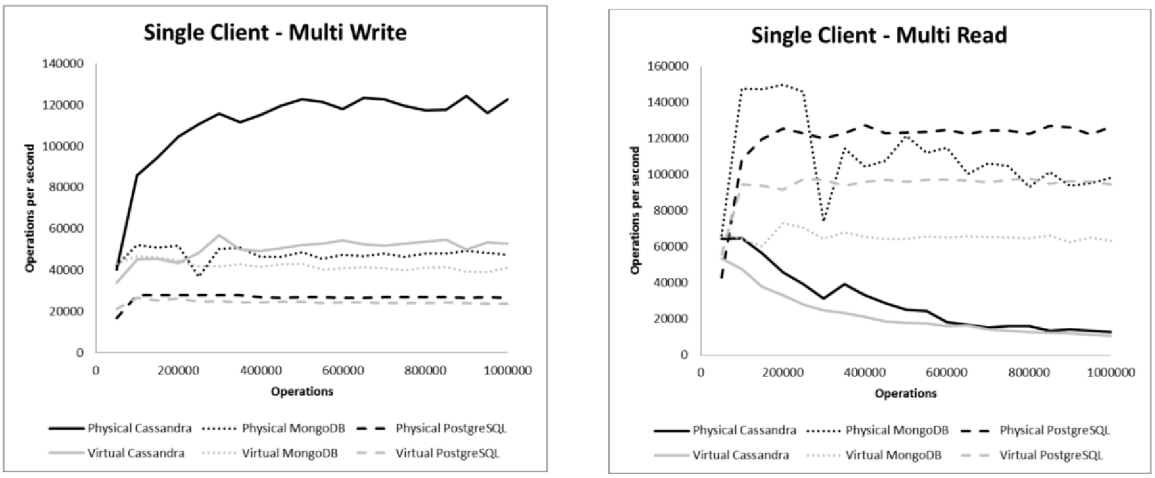
Jedním z hlavních cílů frameworků je vývojářům co nejvíce usnadnit vývoj nových aplikací. Express byl pro tuto práci vybrán kvůli předchozím dobrým zkušenostem s tímto frameworkem. Express je nejrozšířenější JavaScriptový framework s rozsáhlou komunitou vývojářů a dlouhým stabilním vývojem. Pro téměř jakýkoliv problém, který by se mohl vyskytnout při vývoji serverové aplikace, lze nalézt v ekosystému Expresu již existující řešení.

2.2.3 Databázový systém

Databáze je uspořádaný soubor strukturovaných informací nebo dat, obvykle uložených elektronicky v počítačovém systému. Databáze je obvykle řízena databázovým systémem určeným pro správu uložených data a struktury samotné databáze. Data, databázový systém a další aplikace, které jsou s nimi spojeny se často společně označují pouze jako databáze.

Databáze se obecně dělí na relační a nerelační. Relační databáze jsou silně strukturované a používají programovací jazyk SQL. V relační databázi jsou data uspořádána do předem definovaných tabulek s řádky a sloupci. Data mohou mezi sebou mít předem definované vztahy. Například tabulka všech knih v knihovně může obsahovat sloupec s odkazem na autora dané knihy, který je uložený v jiné tabulce se všemi ostatními autory. Relační databáze jsou často volené, protože vývojáři jsou s nimi dobře obeznámeni a kvůli stabilitě těchto databází. Relační databáze se obvykle dobře škálují vertikálním způsobem, tj. rozšířením jednoho serveru, na kterém běží. Hůře se však škálují horizontálním způsobem, tj. přidáváním dalších serverů do clusteru.

Nerelační databáze většinou nepodléhají pevné struktuře a podporují různé datové struktury s flexibilními schématy [65–68]. Veen a kol. [69] uvádějí, že nerelační databáze mají vyšší výkon při zápisu do databáze a jsou vhodné pro škálování. Relační databáze naopak vynikají při čtení z databáze a v případech, kdy potřebujeme vyšší konzistenci dat a flexibilnější databázové dotazy.

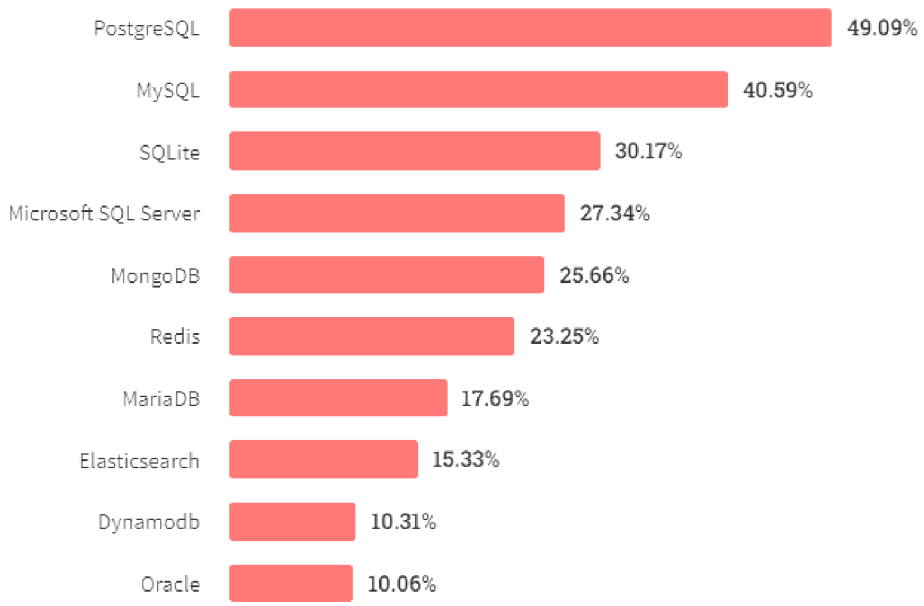


Obr. 11 Porovnání výkonu relačních a nerelačních databází [69]

V následujících podkapitolách jsou podrobněji popsány některé z nejčastěji používaných databází.

2.2.3.1 PostgreSQL

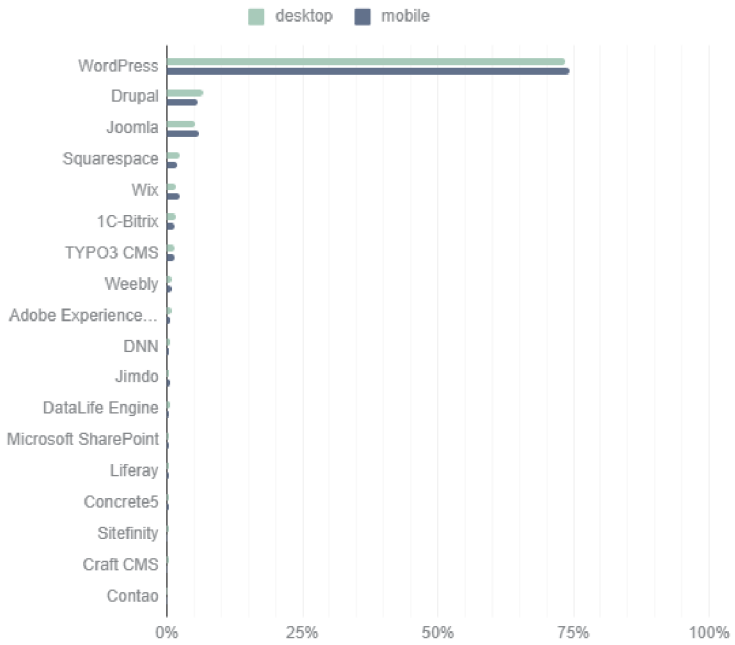
PostgreSQL, také známý jako Postgres, je bezplatný relační databázový systém s otevřeným zdrojovým kódem. Tento databázový systém má za sebou více než dvacet sedm let aktivního vývoje. První verze Postgresu byla vyvinuta v roce 1996 na kalifornské univerzitě v Berkeley a od té doby má tento databázový systém za sebou více než dvacet sedm let aktivního vývoje svojí open-source komunitou. Postupným vývojem se Postgres zařadil mezi špičkové databázové systémy. Hlavními přednostmi Postgresu je jeho popularita, aktivní komunita, kvalitní zdrojový kód a vynikající dokumentace. Postgres se pravidelně umisťuje na prvních místech vývojářských průzkumů jako jeden z nejpoužívanějších a nejoblíbenějších databázových systémů – viz obr. 12. Postgres je primárně vyvíjen pro operační systém Linux, ale je možné jej nainstalovat i na operačním systému Windows, macOS a dalších [26, 70, 71].



Obr. 12 Výsledky průzkumu použití databázových systémů mezi profesionálními vývojáři [27]

2.2.3.2 MySQL

MySQL je jeden z nejznámějších relačních databázových systémů. Podobně jako Postgres je i MySQL open-source a volně šiřitelný. Díky své výkonnosti, spolehlivosti a jednoduchosti se MySQL stal oblíbeným databázovým systémem používaným ve spojení s redakčním systému WordPress, který je do dnešního dne nejpopulárnější redakční systém (viz obr. 13) a v současné době ho využívá přes 40 % veškerých webových stránek [72]. Databázový systém MySQL je často používaný ale i mimo redakční systém WordPress společnostmi jako je např. Facebook, Twitter, YouTube a mnoho dalších. MySQL aktuálně spadá pod společnost Oracle, která řídí jeho vývoj. MySQL je stále open-source software a jeho GPL licence povoluje modifikaci a šíření zdrojového kódu, což umožnilo vznik databázového systému MariaDB, jakožto větev MySQL, z obav původních vývojářů o směru vývoje po převzetí MySQL společností Oracle [73, 74].



Obr. 13 Procentuální podíl použití redakčních systémů [74]

2.2.3.3 MongoDB

MongoDB je nerelační (NoSQL) databázový systém s otevřeným zdrojovým kódem vyvíjený společností MongoDB Inc. od roku 2007. Data jsou ukládána pomocí jednoduchých párů klíč / hodnota, jako dokumenty podobné JSONu. Data nepotřebují a ani nemají pevně definované schéma, jako SQL, což usnadňuje ukládání nestrukturovaných dat. Absence jednotného schématu ale může znesnadnit zpětné získávání dat, kdy nemáme žádnou garanci o typu ani dostupnosti dat. Velkou předností MongoDB je, že může běžet na několika serverech najednou, což umožňuje horizontální škálovatelnost pro rozprostření zátěže serverů a rychlé databázové dotazy. MongoDB je tak vhodný databázový systém pro velké objemy dat s nutností vysoké dostupnosti. MongoDB je dostupný jako služba poskytovaná společností MongoDB Inc., kde zákazníci mohou využívat plně spravované databázové služby. Od roku 2018 je MongoDB vydávaný pod SSPL licencí, což znamená, že společnosti, které chtějí provozovat tento databázový systém jako samostatnou zpoplatněnou službu musí buď získat licenci od MongoDB Inc., nebo otevřít zdrojový kód své služby pro komunitu. Některé organizace toto považují za porušení tradiční open-source filozofie. Ve všech ostatních případech ale lze MongoDB bezplatně provozovat na vlastním serveru, což umožňuje uživatelům nainstalovat a spravovat databázi podle vlastních potřeba a preferencí, podobně jako ostatní databázové systémy [68, 75].

2.2.3.4 SQLite

SQLite je knihovna implementující samostatný, bezserverový SQL databázový systém s nulovou konfigurací. Zdrojový kód je šíře pod licencí public domain a je tak bezplatně dostupný pro jakékoliv účely. Na rozdíl od většiny ostatních SQL databází nemá SQLite samostatný serverový proces. SQLite čte a zapisuje přímo do běžných souborů na disku. Kompletní databáze s mnoha tabulkami je obsažena v jediném souboru na disku. Samostatná knihovna může být menší než 750 KiB. SQLite má reputaci jako velmi spolehlivý nástroj a každá nová verze je podrobena miliónům automatizovaným testům, které zajišťují 100% pokrytí zdrojového kódu testy. SQLite je optimalizovaný pro velký objem čtení, avšak umožňuje pouze jeden zápis do databáze. SQLite je tak vhodný pro lokální správu dat, ale nelze ho škálovat tak dobře, jako jiné tradiční databázové systémy [76].

SQLite je nejvíce používaný databázový systém na světě. Několik instancí SQLite lze nalézt v každém smartphonu s operačním systémem Android a iOS, na každém operačním systému Windows a Mac OS-X, ve většině webových prohlížečů a na mnoho dalších místech. Přesná čísla je nemožné získat, ale SQLite je s velkou pravděpodobností jeden z nejrozšířenějších softwarů společně s knihovnou zlib. SQLite například používá i letový software v letadlech Airbus A350 [77].

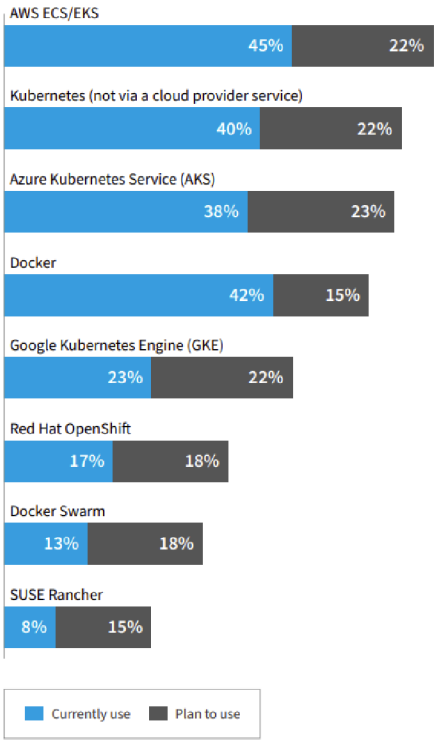
2.2.3.5 Shrnutí databázových systémů

Pro tuto práci byl jako databázový systém vybrán PostgreSQL. Očekávaný objem čtení a zápisů do databáze je relativně malý a všechny výše zmíněné databáze jsou z hlediska výkonu dostačující. Nasbíraná data, která budou ukládána do databáze, jsou silně strukturovaná, což je ideální pro SQL databáze. Aplikace budou přepsány, ale data jsou věčná. Je tedy důležité udržovat data konzistentní a dobře strukturovaná. PostgreSQL společně s MySQL jsou nejpoužívanější databáze na světě a jsou vyvíjeny jako open-source software od roku 1996, resp. 1995. Mají za sebou tak tisíce lidských let vývoje a úsilí, které z nich činí jeden z nejvíce spolehlivých softwarů, co v současné době existuje.

2.2.4 Docker a kontejnery

Docker je softwarová platforma s otevřeným zdrojovým kódem, která umožňuje vývojářům spravovat kontejnery – lehké standardizované jednotky, které obsahují vše, co software potřebuje ke svému běhu, včetně knihoven, systémových nástrojů, kódu a běhového prostředí. Kontejnery jsou izolované a neznají operační systém na kterém Docker běží a ani jeho soubory. Kontejnerizovaný software běží vždy stejně bez ohledu na prostředí kde je nasazený, na rozdíl od běžného softwaru, který většinou vyžaduje specifické prostředí a podpůrný software. Kontejnerizované aplikace lze tedy provozovat jak v klasických datových centrech, tak i na všech možných platformách poskytovatelů cloudových služeb. Tohoto se často využívá v architektuře microservices (mikroslužby), kdy se monolitní aplikace rozdělí na menší, nezávislé a samostatné služby. Tyto mikroslužby, je možné kontejnerizovat a lze je nasadit kdekoliv a v měřítku, které je v dané chvíli potřeba pro danou část aplikace, což umožňuje velmi dobrou škálovatelnost [78–81].

Kontejnery zjednodušují vývoj a nasazování aplikací a jsou stále populárnější, protože organizace přecházejí na vývoj v cloudu. Docker a kontejnerové služby zaznamenaly velmi rapidní rozšíření a v posledních několika letech dosáhly obrovského úspěchu. Z téměř neznáme technologie se od roku 2013 Docker vyvinul ve standardizované běhové prostředí, které je nyní oficiálně využíváno a podporováno prakticky každým poskytovatelem cloudových služeb [82, 83].



Obr. 14 Výsledky průzkumu nejpoužívanějších kontejnerových nástrojů [83]

Dalším důvodem pro popularitu kontejnerů je, že jsou vhodné pro CI/CD (kontinuální integrace / kontinuální nasazení). Jedná se o metodiku DevOps, která má vývojáře přimět k časté integraci kódu do sdíleného úložiště a k jeho rychlému a častému nasazení. Vývojáři mohou snadno a rychle izolovat kód do soběstačného kontejneru, který může běžet prakticky kdekoliv. Rozsáhlé vývojové projekty lze takto rozdělit mezi více menších agilních týmů a dodávání nového softwaru v kontejnerech lze automatizovat pomocí CI/CD programů jako je např. Jenkins [82].

Díky širokému rozšíření kontejnerů se standardem pro nasazení a provoz kontejnerových aplikací stal software Kubernetes, který je zaměřen na správu životního cyklu kontejnerů. Běžně se využívá k automatické orchestraci stovek kontejnerů běžících na více serverech a dokáže rychle škálovat zdroje kontejnerů a vyrovnávat tak zátěž. V závislosti na poptávce přidává, odstraňuje a monitoruje počty a stav kontejnerů, což usnadňuje nasazování a správu složitých distribuovaných aplikací. Kubernetes byl původně vyvinut společností Google a v roce 2014 byl vydán jako open-source. Kubernetes staví na 15 letech provozování kontejnerů ve společnosti Google a mnoha příspěvcích své open-source komunity [82, 84].

2.3 Hardware

V této kapitole jsou popsány hardwarové součásti meteostanice. Jsou zde blíže probrány současně používané mikrokontroléry a vývojové desky. Na závěr jsou zde popsány některé senzory nutné pro měření meteorologických veličin.

2.3.1 Mikrokontroléry a vývojové desky

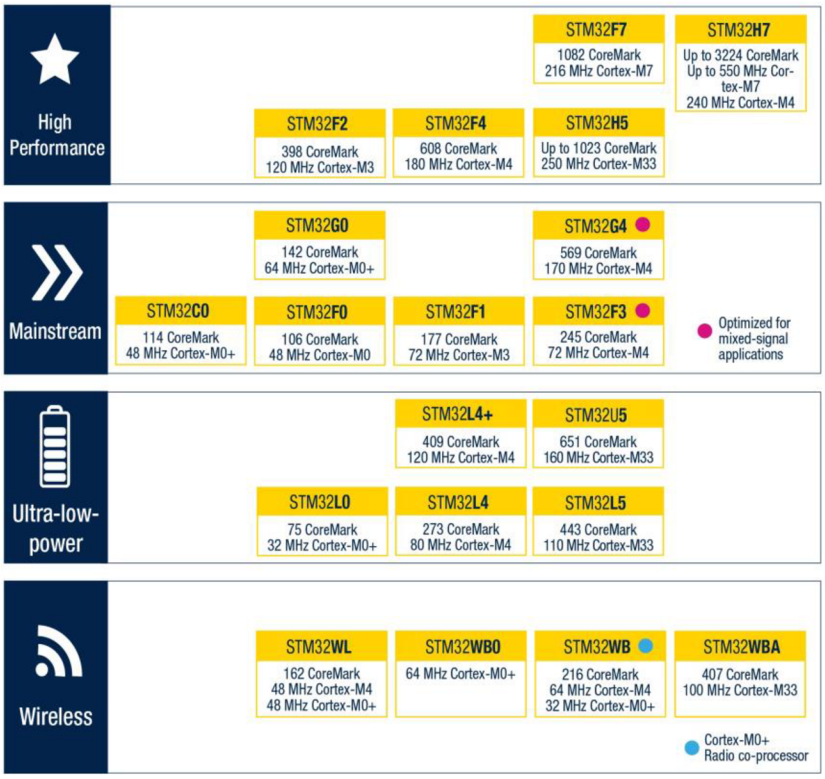
Vývojové desky jsou hardwarové platformy s mikrokontrolérem a dalšími periferními součástkami, které jsou potřeba k jeho funkci. Tyto desky značně usnadňují vývoj nových zařízení a umožňují otestování funkcí před vytvořením finální desky plošných spojů. Vývojové desky jsou určeny jak pro amatéry ke vzdělávacím účelům, experimentování a vytváření rychlých prototypů, tak i pro profesionály pro rychlé ověření konceptu v pracovních podmínkách. K deskám lze velmi snadno připojit různé součásti, které pak lze ovládat pomocí mikrokontroléru nebo třeba také číst různé údaje z připojených senzorů. Výrobci vývojových desek většinou poskytují předpřipravené softwarové knihovny (SDK), ukázkový kód a často i integrovaná vývojová prostředí (IDE) přímo určené pro konkrétní typ desky, což značně usnadňuje a urychluje jejich programování.

V následujících podkapitolách jsou blíže popsány některé z nejpoužívanějších vývojových desek jak v průmyslu, tak v hobby sféře.

2.3.1.1 STM32

V současné době je platforma mikrokontrolérů STM32 od evropské společnosti STMicroelectronics jednou z nejvíce rozšířených na světě. Jsou založeny na procesoru Arm Cotex-M a dodávají se v různých cenových kategoriích s různým výpočetním výkonem a funkcemi. Tyto mikrokontroléry si získaly oblibu v různých odvětvích, jako je automobilový průmysl, zdravotnictví, domácí automatizace a průmyslová automatizace, především proto, že jsou energeticky úsporné, nabízejí vysoký výkon a jsou velmi flexibilní [85].

Mikrokontroléry STM32 jsou vyráběny v 20 různých řadách. Tyto řady lze seskupit do 4 skupin podle účelu použití. První skupinou je na High Performance neboli třída vysoce výkonných miktrokontrolérů. Dále pak Ultra-low-power neboli mikrokontroléry se super nízkou spotřebou energie. Wireless pro bezdrátovou komunikaci a Mainstream neboli hlavní linie mikrokontrolérů pro běžné použití s příznivější cenou [86].



Obr. 15 Hlavní skupiny mikrokontrolérů STM32 [86]

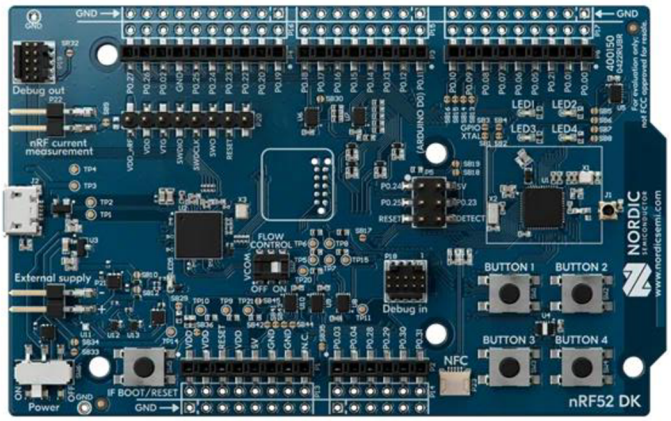
Jednou z nejoblíbenějších vývojových desek pro STM32 jsou oficiální desky Nucleo. Tyto desky lze snadno rozšířit a jsou kompatibilní s mnoha rozšiřujícími hardwarovými moduly pro Arduino. Desky Nucleo mají zabudovaný debugger STLINK, který značně zjednodušuje nahrávání a ladění programů přes USB port. Společnost STMicroelectronics nabízí mnoho variant desky Nucleo, které se dělí do stejných čtyř kategorií jako samotné mikrokontroléry STM32. Ceny těchto desek se pohybují podle modelu, použitého mikrokontroléru a množství funkcí od přibližně 300 Kč až do 1600 Kč [87].

Obr. 16 Vývojová deska Nucleo STM32F070RB [88]

2.3.1.2 Nordic

Norská společnost Nordic Semiconductor je jednou z předních firem v oblasti mikrokontrolérů pro bezdrátovou komunikaci pomocí Bluetooth s velmi nízkou spotřebou energie. Jejich hlavním produktem je řada Bluetooth mikrokontrolérů nRF52, které patří mezi jedny z nejpoužívanějších mikrokontrolérů pro všechna možná Bluetooth zařízení. Používají se například pro bezdrátová sluchátka, příslušenství pro mobilní telefony, bezdrátové myši a klávesnice, zařízení inteligentní domácnosti a mnoho dalších bezdrátových zařízení vyžadující nízkou spotřebu energie. Mimo hlavní sérii nRF52 nabízí Nordic od roku 2018 sérii nRF91 pro rádiové aplikace krátkého dosahu a mobilní sítě LTE a nově od roku 2022 sérii doprovodných čipů nRF70 pro Wi-Fi připojení stávajících produktů Nordic. Všechny tyto řady jsou založeny na procesoru Arm Cotex-M, podobně jako STM32 [89, 90].

Společnost Nordic nabízí oficiální vývojové desky pro všechny hlavní série. Cena vývojové desky s nRF52 pro Bluetooth aplikace je přibližně 900 Kč. Cena desky s nRF91 pro mobilní sítě je přibližně 3700 Kč. Cena desky založené na nRF53 s doprovodným čipem nRF70 pro aplikace vyžadující Wi-Fi připojení je přibližně 1300 Kč [91–93].

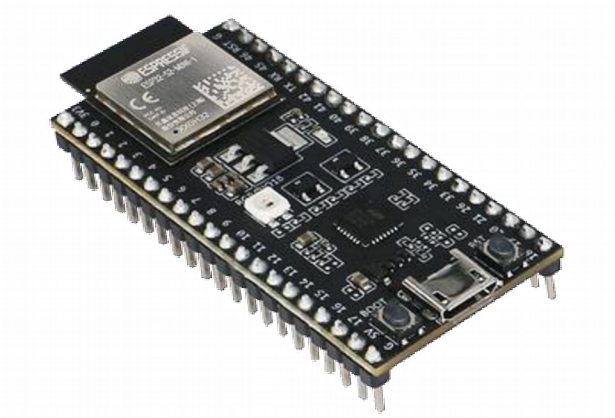


Obr. 17 Vývojová deska nRF52 DK [91]

2.3.1.3 ESP32

ESP32 je řada levných mikrokontrolérů od čínské společnosti Espressif Systems. Jedná se o nástupce předchozí řady mikrokontrolérů ESP8266. Tento mikrokontrolér disponuje téměř všemi existujícími funkcemi ostatních mikrokontrolérů – dvě jádra, Wi-Fi, Bluetooth atd. K programování téměř všech ostatních mikrokontrolérů jsou programátoři většinou nuceni používat IDE od daného výrobce, které jsou často velmi zastaralé. Pro programování ESP32 lze používat moderní IDE jako jsou Visual Studio Code, nebo CLion. Zároveň většina vývojových desek s ESP32 má integrovaný debugger. Mikrokontrolér ESP32 podporuje několik režimů úspory energie. Při aktivním používání všech modulů (Bluetooth, Wi-Fi, RTC, …) odebírá mikrokontrolér 160 až 260 mA. Dále je možné si vybrat ze čtyř režimů spánku, které postupně vypínají komunikační moduly, paměť, hlavní procesor a nakonec i koprocesor. V nejúspornějším režimu hibernace, kdy běží pouze RTC časovač a pár RTC GPIO pinů, klesá odběr na pouhých 2,5 μA [94, 95].

Vývojové desky založené na ESP32 jsou velmi populární v hobby sféře hlavně díky své nízké ceně, která začíná přibližně na 110 Kč. Tyto desky mají více něž dostatečný výkon pro většinu začátečnických IoT projektů. Zároveň mají většinu potřebných funkcí, jako je Wi-Fi, Bluetooth, debugger, režim nízké spotřeby a mnoho dalších. Programování těchto desek je také velmi jednoduché, jelikož pro jednoduché projekty lze použít knihovny Arduino a zkušenější uživatelé mohou využít oficiálních knihoven ESP-IDF, které poskytují mnohem větší kontrolu.



Obr. 18 Vývojová deska ESP32-S2-DevKitM-1 [96]

2.3.1.4 Shrnutí mikrokontroléru

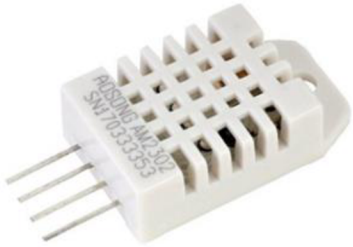
V oblasti embeded systémů se staly mikrokontroléry STM32 v podstatě ekvivalentem 64bitových procesorů Intel. Tyto mikrokontroléry jsou známe pro svou všestrannost a nabízejí řešení pro většinu možných aplikací od zařízení s velmi nízkou spotřebou energie až po vysoce výkonná a výpočetně náročná zařízení. Mikrokontroléry Nordic mají vedoucí postavení v oblasti Bluetooth Low Energy a mezi jejich hlavní výhody patří jejich účinnost pro aplikace vyžadující nízkou spotřebu a Bluetooth konektivitu. ESP32 je z těchto mikrokontrolérů cenově nejdostupnější a i z hlediska objednání jsou vývojové desky s ESP32 nejvíce dostupné. Jejich široká obliba v hobby sféře je také dána kombinací výkonného procesoru, integrovaného Bluetooth a Wi-Fi modulu v jednom cenově velmi výhodném balení. Programování ESP32 je velmi snadné díky možnosti použití moderních IDE a knihoven Arduino a ESP-IDF. Z těchto důvodů byla pro tuto práci vybrána vývojová deska s mikrokontrolérem ESP32.

2.3.2 Senzory

V této kapitole jsou blíže popsány některé senzory, na které bylo nahlíženo pro použití v této práci.

2.3.2.1 DHT22

Senzor DHT22 od společnosti AOSONG je přesnějším nástupcem předchozí verze senzoru DHT11 s větším měřícím rozsahem. Tento senzor se skládá z NTC termistoru pro měření teploty vzduchu a z kapacitního čidla pro měření relativní vlhkosti vzduchu. Rozsah měření relativní vlhkosti je 0 až 100 % s rozlišením 0,1 % a přesností ±2 % (maximálně ±5 %). Rozsah měření teploty je -40 až +80 °C s rozlišením 0,1 °C a přesností ±0,5 °C. Tyto senzory jsou teplotně kompenzovány a kalibrovány v přesné kalibrační komoře a výsledný kalibrační koeficient je uložen do vnitřní paměti. Napájení senzoru je možné od 3,3 do 6 V. Výstupní data jsou přenášena přes jeden vodič v číslicovém tvaru. Cena těchto senzorů v českých obchodech začíná přibližně na 130 Kč [97].



Obr. 19 DHT22 [98]

2.3.2.2 AM2301

Senzor AM2301 je od stejné společnosti jako senzor DHT22 a má tak velmi podobné vlastnosti. Tento senzor má také kapacitní čidlo pro měření relativní vlhkosti vzduchu v rozsahu 0 až 99,9 % s rozlišením 0,1 & a přesností ±3 %. Zároveň má senzor pro měření teploty vzduchu v rozsahu -40 až +80 °C s rozlišením 0,1 °C a přesností ±0,3 °C (maximálně ±1 °C). Napájení senzoru je v rozmezí od 3,3 do 5,2 V. Výstupní data jsou přenášena stejně jako u DHT22 přes jeden vodič v číslicovém tvaru. Hlavním rozdílem tohoto senzoru oproti DHT22 je jeho dvojité krytí a vývody přes delší izolované vodiče. Díky tomu je cena tohoto senzoru přibližně 180 Kč [99].

Obr. 20 AM2301 [100]

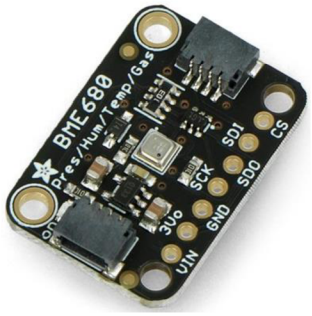
2.3.2.3 BMP280

BMP280 od společnosti Bosch Sensortec je velmi přesný piezo-rezistivní senzor absolutního atmosférického tlaku a teploty. Rozsah měření tlaku je 300 až 1100 hPa s přesností až ±0,12 hPa a rozlišením 0,18 Pa. Díky této vysoké přesnosti lze senzor také použít pro detekci nadmořské výšky s přesností ±1 m. Senzor umožňuje i měření teploty, které ale nedosahuje takové přesnosti. Rozsah měření teploty je 0 až +65 °C s přesností ±0,5 °C a rozlišením 0,01 °C. Senzor vyžaduje napájení 1,71 až 3,6 V. Naměřená data jsou přenášena pomocí sběrnice I^2C, nebo SPI. Cena tohoto senzoru je přibližně 50 Kč [101].

Obr. 21 BMP280 [102]

2.3.2.4 BME680

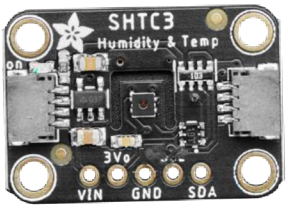
BME680 je senzor umožňující měření teploty, tlaku a vlhkosti vzduchu. Jeho součástí je i MOX senzor, který mění odpor v závislosti na přítomnosti těkavých organických sloučenin ve vzduchu, což umožňuje detekci plynů jako je např. aceton, ethan, ethanol, oxid uhelnatý a další. Senzor teploty měří v rozsahu od -40 do +85 C° s přesností ±1 °C. Senzor relativní vlhkosti měří v plném rozsahu 0 až 100 % s přesností ±3 % a senzor tlaku měří v rozsahu 300 až 1100 hPa s přesností až ±0,12 hPa. Napájení senzoru je v rozmezí 1,2 až 3,6 V a výstupní data jsou přenášena přes sběrnice I^2C, nebo SPI podobně jako u BMP280. Tento senzor kombinuje mnoho funkcí do jediného malého čipu, díky čemuž je cena těchto senzorů přibližně 380 Kč [103].



Obr. 22 BME680 [104]

2.3.2.5 SHTC3

SHTC3 je senzor teploty a relativní vlhkosti od společnosti Sensirion. Kapacitní čidlo umožňuje měření relativní vlhkosti v plném rozsahu od 0 do 100 % s přesností až ±2 % a rozlišením 0,01 %. Teplotní senzor umožňuje měření v rozsahu -40 až +125 °C s přesností ±0,2 °C a rozlišením 0,01 °C. Potřebné napájení je v rozmezí 1,62 až 3,6 V. Naměřená data jsou v digitální formě přenášena přes sběrnici I^2C. Cena tohoto senzoru je přibližně 150 Kč, nicméně tyto senzory nejsou tak rozšířené jako např. DHT22 nebo BMP280 [105].



Obr. 23 SHTC3 [106]

2.3.2.6 Shrnutí senzorů

Pro tuto práci byl vybrán senzor teploty a relativní vlhkosti DHT22 a senzor atmosférického tlaku a teploty BMP280. Hlavními důvody pro výběr těchto senzorů byla jejich rozšířenost v obchodech, nízká cena, dostačující přesnost a dostupnost knihoven pro ESP32.

3 Praktická část

Cílem této práce bylo vytvořit meteostanici, která by měla být schopna dlouhodobě zaznamenávat data a vizualizovat je pomocí online aplikace. Hardwarová část by měla pomocí senzorů snímat environmentální veličiny a následně tyto veličiny zaznamenávat do databázového systému běžícím na serveru. Klientská část by měla být implementována jako webová aplikace, která si může vyžádat zaznamenaná data ze serveru pro vizualizaci časových průběhů environmentálních veličin.

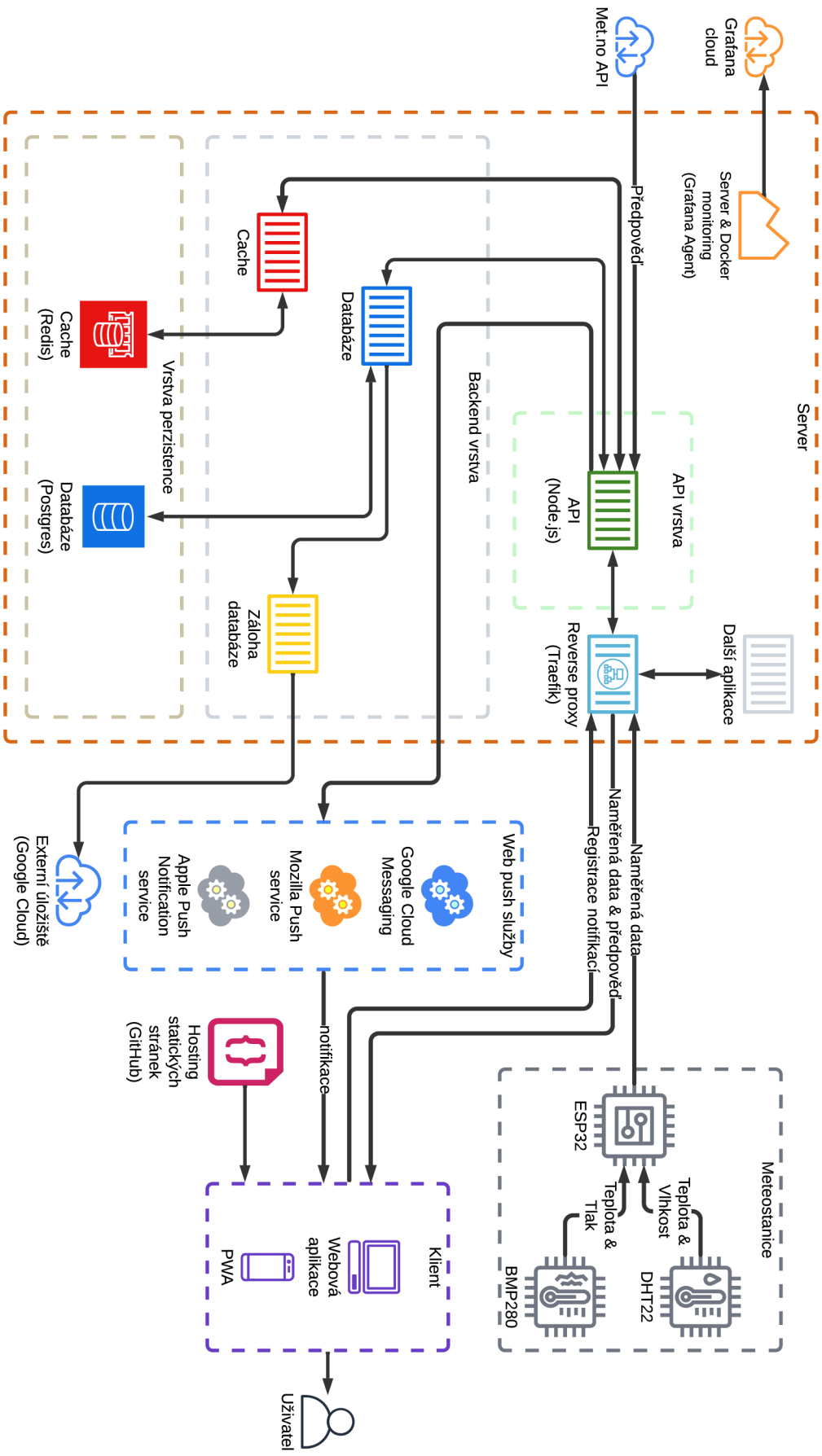
Pro zjednodušenou vizualizaci toku dat celým systémem byl vytvořen diagram zobrazující hlavní komponenty meteostanice a jejich vzájemné interakce – viz obr. 24. Pro ilustraci je zde stručně popsán tok dat hlavními částmi meteostanice během ukládání zaznamenaných hodnot z fyzické meteostanice:

- Mikrokontrolér zaznamenává meteorologické hodnoty ze senzorů
- Zaznamenaná data jsou odesílána na server meteostanice přes Wi-Fi rozhraní
- Server směřuje požadavek z mikrokontroléru na hlavní API meteostanice
- API ukládá nová data do databáze a mezipaměti

Čtení zaznamenaných dat z meteostanice probíhá podobným způsobem:

- Uživatel stahuje statické soubory webové aplikace z webhostingu do svého prohlížeče
- Webová aplikace žádá server o zaznamenaná data
- Server směřuje požadavek na hlavní API meteostanice
- API získává data z mezipaměti, nebo případně z databáze
- API vrací webové aplikaci zaznamenaná data
- Zaznamenaná data jsou zobrazena uživateli v přehledném formátu

Mimo prosté ukládání a čtení dat má server další funkce jako je např. získávání předpovědi, zálohování obsahu databáze, rozesílání notifikací klientům, monitoring serveru, možnost souběžně provozovat další aplikace na serveru a další funkce. V následujících podkapitolách je detailněji popsána implementace jednotlivých částí meteostanice.



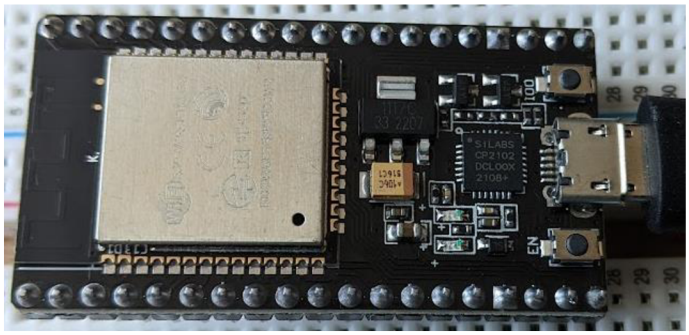
Obr. 24 Diagram hlavních částí a funkcí meteostanice

3.1 Hardware

V této části práce je popsána fyzická část meteostanice a všechny hardwarové komponenty zodpovědné za sběr meteorologických dat. Zároveň je v této kapitole popsán řídící kód mikrokontroléru zodpovědný za ovládání senzorů a všech dalších hardwarových komponent.

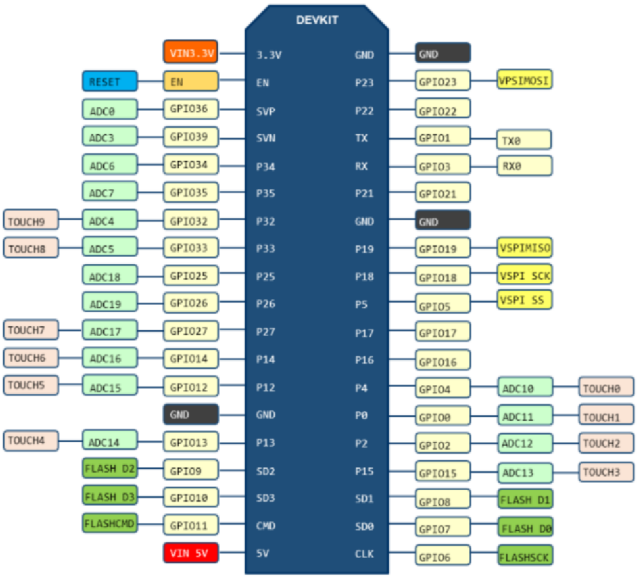
3.1.1 Zapojení elektroniky

Z důvodů popsaných v teoretické části této práce byla pro fyzickou část meteostanice vybrána vývojová deska NodeMCU ESP-32S (v1.1) s mikrokontrolérem ESP32. Dále byl také použit senzor teploty vzduchu a vlhkosti vzduchu DHT22 a senzor atmosférického tlaku a teploty vzduchu BMP280.



Obr. 25 Vývojová deska NodeMCU ESP-32S

Mikrokontrolér ESP32 vyžaduje napájení napětím 3.3 V. Vývojovou desku lze napájet buď napětím 3.3 V na pinu VIN3.3V, napětím 5 V na pinu VIN5V, které je sníženo vestavěným LDO regulátorem napětí na 3.3 V anebo napětím 5 V pomocí USB-A portu. Pro tuto práci byl pro napájení použit USB port, který zároveň umožňuje programování samotného mikrokontroléru pomocí připojeného počítače.



Obr. 26 Piny vývojové desky NodeMCU ESP-32S [107]

Senzor BMP280 byl zapojen přes standardní rozhraní I2C pinem SCL (serial clock) do odpovídajícího pinu mikrokontroléru GPIO22 a pinem SDA (serial data) do odpovídajícího pinu GPIO 21. Senzor dle specifikace výrobce vyžaduje napájení 1,71 V až 3,6 V. Pro napájení senzoru byl použit pin mikrokontroléru VIN3.3V připojený na pin senzoru VCC a zem senzoru a mikrokontroléru GND.

Senzor DHT22 byl zapojen datovým pinem SDA do digitálního výstupu mikrokontroléru na pinu GPIO 14. Senzor dle specifikace výrobce vyžaduje napájení 3,3 V až 6 V. Podobně jako u senzoru BMP280 byly i zde pro napájení použity piny mikrokontroléru VIN3.3V a GND. Pro odstranění rušení byl mezi napájecí a data pin připojen 10 kΩ pull-up rezistor.

3.1.2 Program mikrokontroléru

Hlavní řídící program mikrokontroléru byl vytvořen v programovacím jazyce C++ a frameworku PlatformIO. Výsledný zkompilovaný kód je do mikrokontroléru nahráván přes USB port vývojové desky. Program mikrokontroléru má tři hlavní části, které jsou blíže popsány v této kapitole.

Prvním krokem je inicializace připojených senzorů. Nejprve je připraven pro čtení senzor BMP280. Registry senzoru jsou nastaveny na komunikaci přes I2C a je ověřeno zda

ID senzoru má správnou hodnotu. Ze senzoru jsou přečteny kalibrační koeficienty a parametry vzorkování jsou zapsány do registrů senzoru. Pokud nastavení senzoru BMP280 proběhlo úspěšně, tak je dále inicializován senzor DHT22. Nastavení tohoto senzoru probíhá obdobným způsobem. Pro nastavení obou senzorů jsou použity volně dostupné knihovny od společnosti Adafruit.

```
float BMP_Temperature;
float BMP_Pressure;
float DHT_Temperature;
float DHT_Humidity;

if (bmp.takeForcedMeasurement())
{
    BMP_Temperature = bmp.readTemperature();
    BMP_Pressure = bmp.readPressure();
}
else
{
    Serial.println("[BMP280] Error reading forced measurement");
}

sensors_event_t event;
dht.temperature().getEvent(&event);
if (isnan(event.temperature))
{
    Serial.println(F("[DHT-22] Error reading temperature"));
}
else
{
    DHT_Temperature = event.temperature;
}

dht.humidity().getEvent(&event);
if (isnan(event.relative_humidity))
{
    Serial.println(F("[DHT-22] Error reading humidity"));
}
else
{
    DHT_Humidity = event.relative_humidity;
}
```

Zdrojový kód 1 Čtení hodnot ze senzorů (zkráceno)

Druhým krokem programu mikrokontroléru je změření a čtení hodnot ze senzorů. Všechny naměřené hodnoty jsou uloženy do proměnných, se kterými je možné dále manipulovat. Tato část programu se nachází v hlavní *loop* smyčce a je tak možné čtení provádět kontinuálně.

```
void sendData(float BMP_Temperature, float BMP_Pressure, float DHT_Temperature,
float DHT_Humidity)
{
    HTTPClient http;
    String url = "https://weatherapi.bladesheng.com/api/readings";

    http.begin(url);

    http.addHeader("Content-Type", "application/json");
    http.addHeader("password", apiPassword);
    http.addHeader("short", "true");

    StaticJsonDocument<128> doc;
    doc["temperature_BMP"] = BMP_Temperature;
    doc["pressure_BMP"] = BMP_Pressure;
    doc["temperature_DHT"] = DHT_Temperature;
    doc["humidity_DHT"] = DHT_Humidity;

    String jsonOutput;
    serializeJson(doc, jsonOutput);

    uint8_t httpCode = http.POST(String(jsonOutput));

    http.end();
    Serial.print("HTTP response code: ");
    Serial.println(httpCode);
}

connectWiFi();
sendData(BMP_Temperature, BMP_Pressure, DHT_Temperature, DHT_Humidity);
disconnectWiFi();

esp_sleep_enable_timer_wakeup(5 * 60 * 1000000);
esp_deep_sleep_start();
```

Zdrojový kód 2 Odeslání naměřených hodnot na server a hibernace (zkráceno)

Posledním krokem programu mikrokontroléru je odeslání naměřených hodnot na server meteostanice. Mikrontrolér je nejprve připojen k přednastavené Wi-Fi síti. Z naměřených dat je vytvořeno tělo požadavku ve formátu JSON. Požadavku jsou nastaveny všechny potřebné hlavičky, včetně hesla potřebného pro ukládání měření na serveru. Požadavek je následně odeslán na server na endpoint */readings* jako POST požadavek. Po odeslání požadavku je mikrokontrolér odpojen od Wi-Fi sítě a je přiveden do nejúspornějšího režimu hibernace, ve kterém je odběr pouhých 2,5 µA. V režimu spánku mikrokontrolér zůstává 5 minut a po jeho ukončení je program mikrokontroléru spuštěn znovu. Mikrokontrolér takto pravidelně každých 5 minut odesílá aktuální data ze senzorů na server kde jsou dále zpracována.

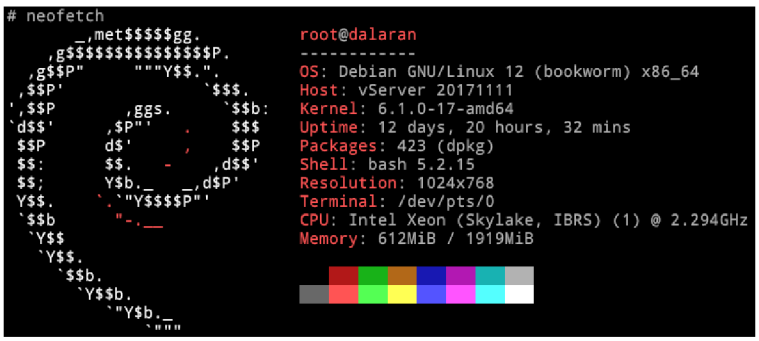
3.2 Server

Server je centrální částí meteostanice zajišťující komunikaci mezi jejími hlavními prvky – fyzická meteostanice sbírající meteorologická data, klient zobrazující data uživateli, databáze pro ukládání dat a další webové služby. V následujících podkapitolách je popsán postup vytvoření serveru potřebného pro provozování všech webových služeb meteostanice. Zároveň jsou v této kapitole popsány všechny hlavní funkce serveru, včetně API, databáze, nasazení serveru a další služby.

3.2.1 Hosting serveru

Pro hosting serveru byla vybrána německá společnost Hetzner. Hetzner je mezinárodně uznávaný poskytovatel webového hostingu provozující vlastní datová centra v Německu, USA a ve Finsku. Hetzner nabízí služby pro sdílený webhosting, spravované servery, virtuální privátní servery (VPS), dedikované servery, úložné servery a mnoho dalších webových služeb.

VPS poskytuje izolované virtualizované prostředí na fyzickém serveru, který je sdílen s ostatními uživateli. Každé prostředí má pevně vyhrazené zdroje, které může využívat pouze daný uživatel. VPS je v mnoha ohledech podobný fyzickému dedikovanému serveru, ale většinou za menší cenu a s méně výkonným procesorem. Nejlevnější VPS, který Hetzner nabízí, je CX11 s jedním vCPU, 2 GB RAM, 20 GB úložného místa na SSD a 20 TB přenosu dat za 3,8 euro měsíčně. Pro účely této práce byly tyto prostředky naprosto dostačující a zbylé zdroje lze zároveň využít pro hostování dalších menších aplikací. Jako operační sytém byla pro server použita současná stabilní verze Debianu – Bookworm (Debian 12). K serveru je přistupováno pomocí standardního protokolu SSH a autentizace je zajištěna přes pár veřejného a privátního klíče.



Obr. 27 Základní vlastnosti VPS

3.2.2 Kontejnerizace

Na server byl nainstalován kompletní balíček Docker Engine, který umožňuje spouštět kontejnerizované aplikace a Docker Compose projekty. Bylo předpokládáno, že všechny aplikace na serveru budou provozovány v Docker kontejnerech. Kontejnery značně usnadňují nasazování aplikací na serveru a zajišťují, že při vývoji aplikace poběží ve stejném prostředí jako na produkčním serveru. Obsahují všechny potřebné závislosti ke spuštění aplikace, takže na server není potřeba instalovat žádný další podpůrný software. Kontejnery je možné provozovat prakticky kdekoliv, takže v případě potřeby lze kontejnerizované aplikace velmi rychle a jednoduše nasadit na jiné servery. Nejsme tak nijak vázáni na určité prostředí jednoho specifického poskytovatele webových služeb.

Instalace Docker Engine na server je velmi jednoduchá. Při postupování podle oficiální dokumentace byl v prvním kroku přidán repozitář s Docker balíčky pro Debian do zdrojů *apt* a v druhém kroku byly nainstalovány všechny potřebné balíčky následujícím příkazem:

```
sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin
```

Zdrojový kód 3 Instalace Dockeru

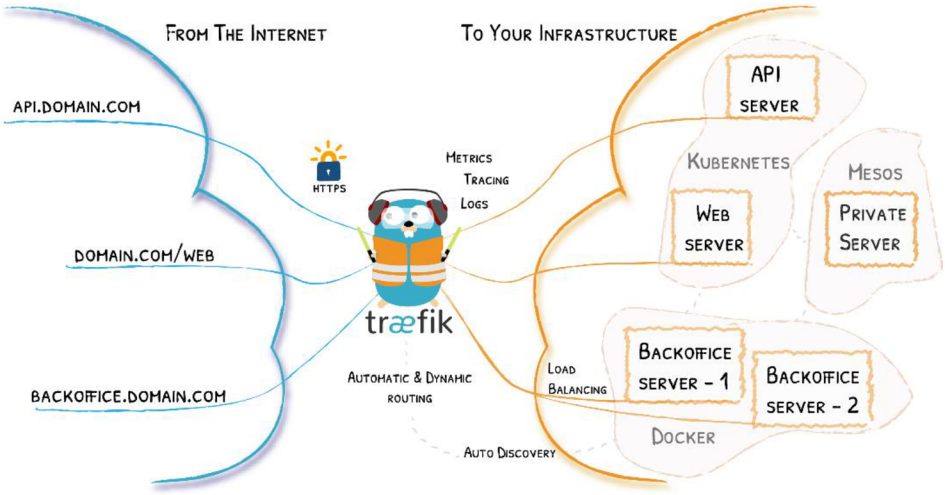
3.2.3 Reverzní proxy

Reverzní proxy se většinou používá jako jakýsi prostředník mezi příchozími požadavky z internetu a několika připojenými servery. Jedním z nejčastějších použití reverzní proxy je pro vyvažování zátěže příchozích požadavků mezi několik serverů zapojených v clusteru. Příchozí požadavky jsou rovnoměrně rozdělovány mezi všechny aktivní servery a je možné přidávat / odebírat servery v závislosti na počtu požadavků a zátěži na serverech. Reverzní proxy však nemusí rozdělovat požadavky pouze mezi samotnými servery, ale může požadavky také směřovat v rámci jednoho serveru s několika běžícími aplikacemi. Aplikace naslouchají na různých portech téhož serveru se stejnou IP adresou a reverzní proxy analyzuje každý příchozí požadavek a doručí jej do správné aplikace. Směřování může být zajištěno na základě subdomény specifikované v požadavku – např. všechny požadavky s adresou *blog.example.com* budou směřovány na jeden webový server a požadavky na *api.example.com* budou směřovány na jiný webový server běžící na stejném počítači. Hlavní výhodou je, že lze takto na jediném serveru provozovat

současně několik webových služeb běžících na jediném sdíleném portu – nejčastěji např. port 80 (HTTP), nebo 443 (HTTPS). Reverzní proxy jsou implementovány ve většině populárních webových serverech jako jsou např. Nginx, Apache a Caddy.

Provozování více webových serverů současně na portu 443 na jednom fyzickém serveru bylo jedním z požadavků této práce, jelikož v současné době i nejlevnější VPS jsou dostatečně výkonné a mají mnohem více zdrojů než bylo odhadováno, že bude potřeba. Jak již bylo zmíněno v předchozí kapitole, bylo předpokládáno, že všechny webové servery a další podpůrné aplikace budou na serveru provozovány v kontejnerech. Pro tento účel byl vybrán Traefik.

Traefik je moderní open-source reverzní proxy a load balancer napsaný v jazyce Go, který splňuje všechny požadavky. Umožňuje směřování příchozích požadavků v rámci jednoho fyzického serveru s několika běžícími webovými servery. Traefik (2016) je relativně nový oproti jiným populárním reverzním proxy jako jsou Nginx (2004) nebo Apache (1995) a již od základu je plně kompatibilní s kontejnery a značně zjednodušuje směřování požadavků do kontejnerizovaných aplikací. Na rozdíl od tradičních staticky konfigurovaných reverzních proxy je Traefik dynamicky konfigurovaný podle kontejnerů samotných, které objevuje např. naslouchánim Docker socketu. Mimo jiné lze Traefik také použít s poskytovatelem ACME, jako je např. *Let's Encrypt* a dokáže již v základu automaticky generovat platné SSL / TLS certifikáty a zajišťovat jejich včasné obnovení.



Obr. 28 Hlavní funkce Traefiku [108]

Traefik je na serveru provozovaný uvnitř kontejneru, který je spouštěn pomocí Docker Compose. V souboru *docker-compose.yaml* byla definována všechna potřebná nastavení kontejneru následovně:

```yaml
services:
  traefik:
    image: traefik:v2.10
    container_name: "traefik"
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - ./traefik.yaml:/traefik.yaml
      - ./traefik_dynamic.yaml:/traefik_dynamic.yaml
      - ./acme.json:/acme.json
      - ./usersfile.txt:/usersfile.txt
    networks:
      - web

networks:
  web:
    external: true
```

Zdrojový kód 4 Nastavení Traefik kontejneru

Mezi hlavní nastavení patří:

image:

- aktuální verze oficiálního obrazu kontejneru Traefiku, která má být použita

ports:

- 80 a 443 – porty, na kterých Traefik přijímá příchozí požadavky

volumes:

- docker socket, který Traefik poslouchá pro automatickou konfiguraci nových kontejnerů
- traefik.yaml a traefik_dynamic.yaml – soubory pro statickou a dynamickou konfiguraci Traefiku
- acme.json – soubor pro ukládání TLS certifikátů a jejich opětovné načtení i po restartování kontejneru

network:

- web – název Docker sítě, ke které musí být připojeny všechny kontejnery, které mají komunikovat s Traefikem. Síť je definována jako externí, aby byla přístupná i pro ostatní Docker Compose projekty

Po nastavení konfiguračních souborů lze tento projekt jednoduše spustit příkazem *docker compose up -d*, který kontejner s Traefikem spustí na pozadí. Pro zpřístupnění nových kontejnerů s webovými servery stačí tyto kontejnery připojit k Docker síti *web* a označit je Docker tagy určující subdoménu daného kontejneru. Traefik novým kontejnerům automaticky generuje TLS certifikáty na 90 dní a obnovuje je 30 dní před jejich vypršením. Webový server je pak dostupný na specifikované subdoméně přes protokol HTTPS bez jakékoliv další nutné konfigurace.

Pro tento Docker Compose projekt byl vytvořen *systemd unit* soubor zodpovědný za automatické spuštění projektu vždy po spuštění serveru.

```
[Unit]
Description=Traefik docker compose
Requires=docker.service
After=docker.service

[Service]
Type=oneshot
RemainAfterExit=yes
WorkingDirectory=/root/repos/vps-traefik/
ExecStart=docker compose up -d
ExecStop=docker compose down

[Install]
WantedBy=multi-user.target
```

Zdrojový kód 5 Systemd unit soubor pro automatické spuštění Traefiku

3.2.4 Databáze

Jedním z cílů této práce je dlouhodobě zaznamenávat naměřená data z meteostanice. Z důvodů popsaných v teoretické části této práce byl pro tento účel vybraný databázový systém Postgres. Postgres je podobně jako další části serveru provozovaný uvnitř kontejneru následovně:

```
services:
  db:
    image: postgres:15.2
    restart: unless-stopped
    volumes:
      - db-data:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}

volumes:
  db-data:
```

Zdrojový kód 6 Nastavení Postgres kontejneru

Postgres kontejner používá aktuální verzi oficiálního Postgres obrazu. Sekce *volumes* zajišťuje, že data zaznamenaná do databáze budou uložena na disk mimo kontejner, aby v případě restartování kontejneru nedošlo ke ztrátě dat. V sekci *environment* jsou kontejneru předány proměnné s přihlašovacími údaji a název výchozí databáze.

Pro Postgres kontejner a zbytek API meteostanice byl vytvořen samostatný Docker Compose projekt. Z bezpečnostních důvodů není Postgres kontejner v základním nastavení nijak přístupný z internetu přes Traefik. Dotazování databáze bude standardně probíhat pouze skrze kontejner s Node API, který je popsaný v následující kapitole.

3.2.5 API pro naměřená data

Hlavní částí serveru je API zajišťující komunikaci mezi všemi částmi projektu meteostanice. Jako jazyk serveru byl vybraný TypeScript hlavně kvůli výhodám, které přináší statické typování do JavaScriptu a také kvůli jeho rozšířenosti. Pro běh zkompilovaného JavaScriptového kódu bylo vybráno běhové prostředí Node.js společně s frameworkem Express. Důvody pro výběr těchto technologií jsou popsány v teoretické části této práce. Podstatou tohoto API je Node.js server, přijímající naměřená meteorologická data, která ukládá do databáze. Server současně přijímá HTTP dotazy od klientů a odpovídá jim daty přečtenými z databáze. Tento server, stejně jako všechny další komponenty, byl provozován uvnitř kontejneru. Základem pro tento kontejner je aktuální verze oficiálního Node obrazu, na které se pomocí Dockerfile sestavuje a spouští server samotný.

3.2.5.1 Server kontejner

Dockerfile serveru je složený ze dvou vrstev nazvané *build* a *deploy*. Jak již název napovídá, v první vrstvě je server sestaven a v druhé vrstvě je server spuštěn. Výsledkem je malý produkční kontejner obsahující pouze soubory potřebné pro běh serveru bez nástrojů potřebných k jeho sestavení.

```
FROM node:20.11.1-bookworm-slim AS build
RUN apt-get update && apt-get upgrade -y
RUN npm -g install pnpm@8.15.5

WORKDIR /app

COPY package.json pnpm-lock.yaml ./
COPY prisma/schema.prisma ./
ENV NODE_ENV production
RUN pnpm install --frozen-lockfile

COPY . .

RUN pnpm run build
```

Zdrojový kód 7 Build vrstva server kontejneru

V první vrstvě jsou staženy a nainstalovány přesné verze všech použitých knihoven, které jsou definovány v souborech *package.json* a *pnpm-lock.yaml*. Následně je do kontejneru zkopírován zdrojový TypeScript kód serveru samotného, který je pak zkompilován do spustitelného JavaScript kódu. Pořadí těchto kroků je důležité hlavně pro lokální vývoj serveru, protože pokud nezměníme použité knihovny, tak Docker si tento krok může uložit do mezipaměti a nemusí ho opakovat při každé změně zdrojového kódu samotného.

```
FROM node:20.11.1-bookworm-slim AS deploy
RUN apt-get update && apt-get upgrade -y
RUN npm -g install pnpm@8.15.5
RUN apt-get install openssl -y
WORKDIR /app

COPY --from=build /app/dist /app/dist
COPY --from=build /app/node_modules /app/node_modules
COPY --from=build /app/package.json /app/package.json
COPY --from=build /app/pnpm-lock.yaml /app/pnpm-lock.yaml
COPY --from=build /app/prisma /app/prisma

ENV NODE_ENV production

CMD pnpm run start
```

Zdrojový kód 8 Deploy vrstva server kontejneru

V druhé vrstvě jsou do kontejneru nainstalovány pouze potřebné systémové závislosti a z předchozího stupně jsou zkopírovány artefakty s výsledným JavaScriptovým kódem, který je potřebný pro běh serveru. Tato vrstva je zakončena příkazem pro spuštění serveru po spuštění kontejneru.

Takto sestavený kontejner lze přidat do hlavního Docker Compose projektu následovně:

```
services:
  server:
    image: ghcr.io/weather-blade/weather-server:latest
    restart: unless-stopped
    depends_on:
      - redis
      - db
    environment:
      PORT: ${PORT}
      API_PASSWORD: ${API_PASSWORD}
      DATABASE_URL: ${DATABASE_URL}
      VAPID_PRIVATE_KEY: ${VAPID_PRIVATE_KEY}
      VAPID_PUBLIC_KEY: ${VAPID_PUBLIC_KEY}

    networks:
      - "web"
      - "default"

    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.weatherapi.rule=Host(`weatherapi.bladesheng.com`)"
      - "traefik.http.routers.weatherapi.entrypoints=websecure"
      - "traefik.http.routers.weatherapi.tls.certresolver=myresolver"
      - "traefik.http.services.weatherapi.loadbalancer.server.port=${PORT}"

networks:
  web:
    external: true
```

Zdrojový kód 9 Nastavení server kontejneru

V sekci *networks* je definováno, že server kontejner je připojený k výchozí Docker síti *default* daného Docker Compose projektu. K této síti je automaticky připojen i kontejner s databází, což umožňuje server kontejneru přístup k databázi. Server kontejner je také připojen k externí síti *web*, na které je i Traefik kontejner. V sekci *labels* je přesně definováno nastavení pro Traefik, podle kterého má být server zpřístupněn zbytku internetu, na jaké doméně atd.

Podobně jako v případě Traefiku byl i pro tento Docker Compose projekt vytvořen *systemd unit* soubor zodpovědný za automatické spuštění projektu vždy po spuštění serveru.

3.2.5.2 Node.js API

V připraveném kontejneru s Node.js je nyní možné spustit samotný webový server. Použitý framework Express velmi silně využívá middleware pro zpracovávání příchozích požadavků. Middleware jsou funkce, které mají přístup k objektu požadavku, objektu odpovědi a k následující funkci v řetězci middlewaru. Middleware funkce může např. upravovat objekty požadavku a odpovědi, ukončit cyklus požadavek-odpověď, nebo zavolat následující middleware funkci. Každý požadavek, který server obdrží, postupně putuje řetězcem middlewaru, dokud některá z middleware funkcí řetězec neukončí odpovědí na požadavek. V hlavním souboru *index.ts* je definováno vytvoření serveru následovně:

```
const app = express();

app.use(morgan("combined"));
app.use(helmet());
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(compression());
app.use(cors);
app.use("/api", ApiRouter);
app.use(ErrorHandler.handleError);
app.use(notFound);

const port = process.env.PORT || 8080;
app.listen(port, () => {
    console.log(`[server] Server is running at http://localhost:${port}`);
});
```

Zdrojový kód 10 Hlavní middleware serveru

Nejprve je vytvořena nová instance Express aplikace, na kterou je poté připojen všechen middleware. Je zde použit např. middleware Morgan pro logování různých událostí na serveru, nebo middleware Helmet, který pomáhá opravit některé běžné bezpečnostní chyby nastavením bezpečných HTTP hlaviček. Ke konci řetězce je připojen routovací middleware sloužící pro směřování požadavků na adresu */api* do *ApiRouter* middlewaru. Na požadavky s jinou adresou je odpovězeno HTTP kódem *404 nenalezeno*.

```
router.use("/readings", ReadingsRouter);
router.use("/forecast", ForecastRouter);
```

Zdrojový kód 11 Routování */api* požadavků

Uvnitř API routeru jsou dále všechny požadavky na adresu */api/readings* přesměrovány do *ReadingsRouter*, jehož cílem je obsloužit všechny požadavky týkající se samotných dat zaznamenaných z meteostanice.

```
router.get("/events", ReadingEventsController.openConnection);

router.get("/range", ...ReadingsController.getTimeRange);
router.get("/month/full", ...ReadingsController.getMonthFull);
router.get("/month/decimated", ...ReadingsController.getMonthDecimated);
router.get("/24h", ...ReadingsController.getLast24h);

router.get("/:id", ...ReadingsController.getReading);
router.post("/", authenticate, ...ReadingsController.postReading);
router.put("/:id", authenticate, ...ReadingsController.upsertReading);
router.delete("/:id", authenticate, ...ReadingsController.deleteReading);
```

Zdrojový kód 12 Routování */api/readings* požadavků

V routeru *ReadingsRouter* jsou definovány jednotlivé endpointy, které požadavky směřují na příslušné metody kontroléru *ReadingsController*. Pro zaznamenaná data z meteostanice jsou zde metody odpovídající základním CRUD operacím (create, read, update, delete). Např. všechny POST požadavky s nově zaznamenanými daty z meteostanice jsou směřovány do metody *postReading*, která vypadá následovně:

```
public static postReading = [
    checkSchema(ReadingsValidation.reading),
    checkSchema(ReadingsValidation.readingDate),

    async (req: Request, res: Response, next: NextFunction) => {
        try {
            const errors = validationResult(req);
            if (!errors.isEmpty()) {
                throw new AppError(400, "Bad request", errors.array());
            }

            const temperature_BMP = parseFloat(req.body.temperature_BMP);
            const temperature_DHT = parseFloat(req.body.temperature_DHT);
            const pressure_BMP = parseFloat(req.body.pressure_BMP);
            const humidity_DHT = parseFloat(req.body.humidity_DHT);

            const createdAt = req.body.createdAt ? new Date(req.body.createdAt)
            : undefined;

            const reading = await ReadingsService.createReading(
                temperature_BMP,
                temperature_DHT,
                pressure_BMP,
                humidity_DHT,
                createdAt
            );

            if (req.headers["short"] === "true") {
                res.sendStatus(200);
            } else {
                res.json(reading);
            }

            ReadingsEventsService.sendReadingToAll(reading);

            const year = reading.createdAt.getUTCFullYear();
            const month = reading.createdAt.getUTCMonth();
            const cacheNameFull = `month-full-${year}-${month}`;
            const cacheNameDecimated = `month-decimated-${year}-${month}`;
            redisClient.del("readings24h");
            redisClient.del(cacheNameFull);
            redisClient.del(cacheNameDecimated);
        } catch (error) {
            next(error);
        }
    },
];
```

Zdrojový kód 13 Kontrolér pro nově zaznamenaná data

Tento kontrolér nejprve provede validaci naměřených dat. Pokud jsou data ve správném formátu, tak jsou předány metodě *createReading* služby *ReadingsService*, která má za úkol tyto data uložit do databáze. Po úspěšném uložení dat je požadavek uzavřen a všem připojeným klientům je poslána nová událost s těmito naměřenými daty. Zároveň je invalidována veškerá mezipaměť v Redisu, kde by se tato nová data měla nacházet.

Podobným způsobem lze data z API i zpětně získat. Např. pro získání jednoho určitého měření slouží metoda *getReading*, která si data vyžádá od služby *ReadingsService*.

```
public static getReading = [
    checkSchema(ReadingsValidation.readingId),

    async (req: Request, res: Response, next: NextFunction) => {
        try {
            const errors = validationResult(req);
            if (!errors.isEmpty()) {
                throw new AppError(400, "Bad request", errors.array());
            }

            const id = parseInt(req.params.id);
            const reading = await ReadingsService.getById(id);
            if (reading === null) {
                throw new AppError(404, "Reading not found");
            }

            res.json(reading);
        } catch (error) {
            next(error);
        }
    },
];
```

Zdrojový kód 14 Kontrolér pro získání určitého záznamu

```
public static async getById(id: number) {
    const reading = await prisma.readings.findUnique({
        where: {
            id: id,
        },
    });

    return reading;
}
```

Zdrojový kód 15 Služba pro získání určitého záznamu

Pro komunikaci s databází je využívána open-source knihovna Prisma. Prisma je ORM (Object-Relational-Mapping) pro Node.js, který značně usnadňuje práci s databází díky abstrahování SQL dotazů a databázových operací do objektově orientovaného přístupu. Modely všech dat pro Prisma jsou definovány v schema souboru. Model pro data zaznamenaná z meteostanice je definovaný následovně:

```
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
  binaryTargets = ["native", "debian-openssl-3.0.x"]
}

model Readings {
  id        Int      @id @default(autoincrement())
  createdAt DateTime @default(now())

  temperature_BMP Float
  temperature_DHT Float
  pressure_BMP    Float
  humidity_DHT    Float
}
```

Zdrojový kód 16 Prisma schema pro zaznamenaná data z meteostanice

Je zde definován typ databáze společně s její adresou a všechny modely, které budou ukládány do databáze. Z těchto modelů Prisma automaticky generuje migrace a klienta se všemi potřebnými metodami pro manipulaci s daty modelů, jejichž vstupy a výstupy jsou při použití TypeScriptu plně otypované.

Součástí routeru *ReadingsRouter* jsou dále i endpointy pro získávání více záznamů najednou. Lze získat všechny záznamy z libovolného časového rozmezí, všechny záznamy za posledních 24 hodin a záznamy z určitého měsíce. Tyto záznamy lze také získat jako zdecimované z původních přibližně 8000 záznamů za měsíc na pouhých 1000 záznamů, což zmenšuje objem přenášených dat a snižuje hadwarové nároky na vizualizaci dat. Data jsou decimována pomocí algoritmu LTTB (Largest-Triangle-Three-Buckets) [109], který velmi efektivně snižuje počet dat s minimální ztrátou informací pro vizualizaci v čárovém grafu.

Poslední částí routeru *ReadingsRouter* je endpoint */events*, který umožňuje otevření SSE (Serve-Sent Events) spojení. SSE umožňuje připojenému klientovi v reálném čase přijímat zprávy ze serveru prostřednictvím dlouhodobě otevřeného HTTP připojení. Na rozdíl od WebSocketu je komunikace přes SSE jednosměrná a zprávy jsou doručovány pouze ze serveru ke klientovi (např. do webového prohlížeče uživatele). SSE spojení je na serveru vytvářeno následovně:

```
    public static openConnection = [
        (req: Request, res: Response, next: NextFunction) => {
            const headers = {
                "Content-Type": "text/event-stream",
                "Connection": "keep-alive",
                "Cache-Control": "no-cache",
            };
            res.writeHead(200, headers);
            const connection = new SSEConnection(res);
            ReadingsEventsService.saveConnection(connection);
        },
    ];
```

Zdrojový kód 17 Vytvoření nového SSE spojení

```
export class SSEConnection {
    public id: string;
    public res: Response;
    private isOpen: boolean;
    private static KEEP_ALIVE_MS = 30_000;

    constructor(res: Response) {
        this.id = crypto.randomUUID();
        this.res = res;
        this.isOpen = true;

        setTimeout(this.keepAlive.bind(this), SSEConnection.KEEP_ALIVE_MS);
        this.res.on("close", this.closeConnection.bind(this));
    }

    private keepAlive() {
        if (!this.isOpen) return;

        try {
            this.res.write(":\n\n");
            this.res.flush();
        } catch (error) {
            console.error(error);
        }

        setTimeout(this.keepAlive.bind(this), SSEConnection.KEEP_ALIVE_MS);
    }

    private closeConnection() {
        this.res.end();
        this.res.socket?.destroy();
        this.isOpen = false;
    }
}
```

Zdrojový kód 18 Třída pro udržování aktivního SSE spojení

Metoda *openConnection* mění typ spojení na typ odpovídající SSE a vytváří novou instanci třídy *SSEConnection*, která udržuje spojení aktivní posíláním prázdných zpráv každých 30 sekund, aby spojení nebylo automaticky ukončeno. Tento objekt je uložen do pole se všemi dalšími aktivními spojeními. Při obdržení nových naměřených hodnot z meteostanice pak server okamžitě zašle všem připojeným klientům zprávu z těmito hodnotami.

3.2.6 Předpověď počasí

Jedním z požadavků na server bylo zajištění předpovědi počasí. Vzhledem k omezenému množství zaznamenaných dat a celkové náročnosti tohoto úkolu bylo rozhodnuto, že pro získání předpovědi bude využito externího API, které zajistí přesnější předpověď, než jakou by bylo možné vytvořit z lokálních dat. Jako poskytovatel předpovědi byl vybrán norský metrologický institut MET. API pro získání předpovědi z MET je dostupné veřejnosti a je zcela bezplatné. Součástí požadavku pro předpověď je zeměpisná šířka, délka a volitelně i nadmořská výška. API vrací odpověď ve formátu JSON s hodinovou předpovědí pro následujících 9 dní podle současného modelu počasí MET. Součástí předpovědi pro každou hodinu jsou předpokládané hodnoty teploty, oblačnosti, rychlosti a směru větru, rosného bodu, srážek, vlhkosti, a další hodnoty. Institut MET má mnoho dalších API, které je možné využít. Jedním z nich je také API pro získání času východu a západu slunce pro dané zeměpisné souřadnice, kterého bylo v této práci také využito.

```
router.get("/", ForecastController.getForecastSunrise);
router.get("/notification", ForecastController.getNotification);
```

Zdrojový kód 19 Endpointy pro předpověď

Na serveru meteostanice vytvořen endpoint */forecast*, který slouží jako zprostředkovatel pro získání předpovědi z API institutu MET. Požadavky pro získání předpovědi je možné posílat i přímo z klienta (webových stránek), což ale porušuje podmínky služeb MET. Na endpointu */forecast* lze získat kompletní předpověď pro všech 9 dní a dále zde byl vytvořen endpoint */forecast/notification* pro získání zkrácené textové verze předpovědi pro aktuální den.

```
public static async getForecast(): Promise<ITimePointForecast[]> {
    if (
        METService.timePointForecast.length > 0 &&
        METService.forecastExpires.getTime() > Date.now()
    ) {
        return METService.timePointForecast;
    }

    const url =
    `https://api.met.no/weatherapi/locationforecast/2.0/compact?&lat=${METServic
    e.latitude}&lon=${METService.longitude}&altitude=${METService.altitude}`;

    const response = await fetch(url, {
        method: "GET",
        headers: {
         "User-Agent": "https://github.com/weather-blade/weather-server
         keadr23@gmail.com",
         If-Modified-Since": METService.lastModified,
        },
    });

    if (response.ok) {
        const responseData = await response.json();
        METService.timePointForecast = responseData.properties.timeseries;
    }

    let expiresHeader = response.headers.get("expires");
    if (expiresHeader === null) {
        console.warn("Expires header is missing", response.headers);
        expiresHeader = new Date(Date.now() + 40 * 60_000).toISOString();
    }
    const randomDelay = UtilFns.randomIntFromInterval(3, 6);
    METService.forecastExpires = new Date(
        new Date(expiresHeader).getTime() + randomDelay * 60_000
    );

    let lastModifiedHeader = response.headers.get("last-modified");
    if (lastModifiedHeader === null) {
        console.warn("Last-modified header is missing", response.headers);
        lastModifiedHeader = new Date().toISOString();
    }
    METService.lastModified = lastModifiedHeader;

    return METService.timePointForecast;
}
```

Zdrojový kód 20 Získání předpovědi z API institutu MET

Požadavky na */forecast* jsou směřovány do kontroléru *ForecastController*, který získává předpověď ze služby *ForecastService*, a ta data získává ze služby *METService* metodou *getForecast*. Tato metoda nejprve zkontroluje, zda již v mezipaměti není uložená platná předpověď. Pokud předpověď není uložená nebo předpovědi vypršela platnost, tak se na MET API pošle požadavek pro získání nové předpovědi. Součástí požadavku musí být hlavička *User-Agent* identifikující server, ze kterého je požadavek zaslán a zároveň pokud je to možné i hlavička *If-Modified-Since* s datem poslední předpovědi, která je uložená v mezipaměti. MET aktualizuje model předpovědi přibližně každých

30 minut a tato předpověď se vždy nemusí nutně změnit. Pokud server MET vyhodnotí, že naše současná předpověď je zastaralá, tak odpoví novou verzí předpovědi. V opačném případě pouze vrátí HTTP status *304 Nezměněno*. V obou případech je nejnovější verze předpovědi uložena do mezipaměti společně s jejími hlavičkami *expires* a *last-modified*. Postup pro získání časů východu a západu slunce je velmi podobný.

3.2.7 Zálohování databáze

Nasbíraná data uložená v databázi je důležité zálohovat pro případ jejich poškození, nebo poškození serveru samotného. Pro účel zálohování databáze byl vytvořen Python skript, jehož úkolem je vytvořit kompletní zálohu databáze a nahrát ji na externí úložiště.

```
process = sh.pg_dump(
    os.getenv("DATABASE_URL"),
    file=FILENAME,
    _out=print,
    _bg=False,
)
```

Zdrojový kód 21 Vytvoření zálohy databáze

```
shutil.make_archive(
    f"{destination_folder}/{zip_name}",
    "zip",
    root_dir,
    base_dir
)
```

Zdrojový kód 22 Vytvoření zip souboru

```
credentials = service_account.Credentials.from_service_account_file(
    './SA_key.json',
    scopes=['https://www.googleapis.com/auth/drive']
)
service = build('drive', 'v3', credentials=credentials)

file_metadata = {
    'name': f'{filename}.zip',
    'parents': [os.getenv("GDRIVE_FOLDER_ID")]
}

media = MediaFileUpload(
    f'./archive/{filename}.zip',
    mimetype='application/zip',
    resumable=True
)

file = service.files().create(
    body=file_metadata,
    media_body=media,
    fields="id, quotaBytesUsed"
).execute()

print(f"File ID: {file.get('id')}")
print(f"File size: {round(int(file.get('quotaBytesUsed')) / 1000000, 2)} MB")
```

Zdrojový kód 23 Nahrání zip souboru na Google Drive

Tento skript pomocí příkazu *pg_dump* extrahuje Postgres databázi do *.sql* souboru a zkomprimuje ho do zip souboru s časovou známkou. Skript se dále autentizuje pomocí Google OAuth a po úspěšném ověření skript nahraje na Google Cloud výsledný zip soubor.

Tento skript je žádané spouštět pravidelně a automaticky. K jeho spouštění byl proto použit cron, což je systémový proces pro automatizované spouštění příkazů v určitý čas, který je běžně dostupný ve většině distribucí Linuxu.

```
0 4 * * * /usr/local/bin/python3 -u /app/weather_backup.py >> /app/cron.log 2>&1
```

Zdrojový kód 24 Cron úloha pro spuštění zálohovacího skriptu databáze

Do souboru crontab byl přidán příkaz pro automatické spuštění tohoto skriptu každý den ve 4 hodiny ráno. Tento příkaz zároveň ukládá veškerý výstup skriptu do souboru *cron.log*.

Python je interpretovaný jazyk a pro spuštění zálohovacího skriptu je potřeba mít nainstalovaný interpret Pythonu. Proto byl vytvořen nový kontejner, jehož jediným účelem je spouštění tohoto skriptu.

```
FROM python:3.12-rc-slim-bookworm
RUN apt-get update && apt-get upgrade -y
RUN apt-get install cron postgresql-client -y

WORKDIR /app
COPY requirements.txt ./
RUN pip install -r requirements.txt
COPY . .
RUN mkdir dump archive

RUN touch /app/cron.log
RUN crontab crontab.txt

CMD printenv > /etc/environment && cron && tail -f /app/cron.log
```

Zdrojový kód 25 Kontejner pro spouštění zálohovacího skriptu

Základem tohoto kontejneru je aktuální verze oficiálního Python obrazu. Do kontejneru je nainstalován cron pro spouštění skriptu a Postgres klient pro komunikaci s databází. Dále jsou do kontejneru nainstalovány všechny potřebné Python knihovny

definované v souboru *requirements.txt* a do kontejneru je zkopírován skript samotný. Sestavení kontejneru je zakončeno příkazem pro zpřístupnění proměnných prostředí pro cron, zapnutí cron démona a sledování výstupu zálohovacího skriptu spuštěného cronem.

Takto sestavený kontejner lze přidat do hlavního Docker Compose projektu následovně:

```
db-backup:
  image: ghcr.io/weather-blade/weather-db-backup:latest
  restart: unless-stopped
  depends_on:
    - db
  environment:
    DATABASE_URL: ${DATABASE_URL}
    GDRIVE_FOLDER_ID: ${GDRIVE_FOLDER_ID}
  volumes:
    - ./SA_key.json:/app/SA_key.json:ro
```

Zdrojový kód 26 Nastavení zálohovacího kontejneru

V sekci *environment* je definována adresa, na které běží databáze uvnitř Docker sítě a ID Google Drive složky, do které budou nahrávány zálohy. V sekci *volumes* je kontejneru zpřístupněn soubor s údaji potřebnými pro autentizaci s Google OAuth.

Výsledkem je kontejner, který automaticky každý den vytváří zálohu databáze a nahrává ji na Google Drive, odkud je snadno přístupná a zálohy zde lze pravidelně kontrolovat.

3.2.8 Nasazení serveru

Pro spuštění výsledného Docker Compose projektu na serveru je potřeba na server dostat sestavené kontejnery, aby je bylo možné spustit.Standardní kontejnery, jako jsou např. kontejnery Traefiku, Postgresu a Redisu je možné přímo stáhnou z veřejného registru Docker Hub a lze je ihned použít. Nestandardní kontejnery pro Node.js API a zálohovací skript je však ale nutné před jejich použitím nejprve sestavit. Pro sestavení kontejnerů byla vybrána služba GitHub Actions, jelikož všechen zdrojový kód pro kontejnery je aktuálně uložen na GitHub repositářích. GitHub Actions umožňuje vytváření opakujících se automatizovaných CI/CD úkolů (akcí), které značně redukují množství ruční práce. Tyto akce jsou vhodné např. právě pro automatizované vytváření kontejnerů v kombinaci s repositářem kontejnerů Github Container Registry (GHCR).

Po nastavení příslušného repositáře je možné vytvořit akci, která bude automaticky budovat a publikovat kontejnery. Akce vytvářející kontejner hlavního Node.js API je definována standardně ve složce *.github/workflows* společně s dalšími GitHub akcemi a vypadá následovně:

```yaml
name: Build and publish Docker image

on:
  push:
    branches: [main]

  workflow_dispatch:

concurrency:
  group: "image-build-publish"
  cancel-in-progress: true

permissions:
  packages: write

jobs:
  job-1:
    name: Build and publish image
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Get Docker tags
        id: docker_tags
        uses: docker/metadata-action@v5
        with:
          images: |
            ghcr.io/weather-blade/weather-server
          tags: |
            type=raw,value=latest,enable={{is_default_branch}}

      - name: Log in to GitHub Container Registry
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Build and push the image
        uses: docker/build-push-action@v5
        with:
          context: .
          file: Dockerfile
          push: true
          tags: ${{ steps.docker_tags.outputs.tags }}
```

Zdrojový kód 27 GitHub akce pro vytvoření kontejneru s hlavním Node.js API

Tato akce má čtyři hlavní kroky. V prvním kroku akce stáhne zdrojový kód z repositáře do svého běhového prostředí. V následujícím kroku je specifikován hlavní tag kontejneru jako *ghcr.io/weather-blade/weather-server*, který slouží pro identifikaci tohoto kontejneru na GHCR. Dále se akce do GHCR přihlásí a v posledním kroku akce vytvoří kontejner ze zdrojového kódu a nahraje ho do registru s tagy specifikovanými v druhém kroku. Tato akce je automaticky spouštěna vždy při změně kódu v hlavní větvi GitHub repositáře a lze ji také spustit manuálně přes stránky GitHubu. Výsledkem je, že na GHCR

je vždy aktuální obraz kontejneru, který server meteostanice může kdykoliv stáhnout a spustit. Postup vytváření kontejneru se zálohovacím skriptem je velmi podobný.

Po připravení všech kontejnerů už stačí na server pouze nahrát samotný Docker Compose projekt. Hlavní *docker-compose.yaml* soubor je verzovaný v gitu jako zbytek projektu, takže stačí pouze stáhnout hlavní repositář příkazem *git clone*. Dále je potřeba na server nahrát soubor s údaji pro autentizaci Google OAuth a vytvořit soubor *.env* s proměnnými prostředí, jako je port serveru, adresa databáze a další. Po vytvoření těchto souborů lze projekt spustit příkazem *docker compose up -d*, který stáhne všechny potřebné kontejnery a projekt spustí na pozadí. Pokud je vše správně nastaveno, tak Traefik automaticky detekuje tento nový projekt a všechny požadavky na subdoménu *weatherapi* budou směřovány do tohoto projektu.

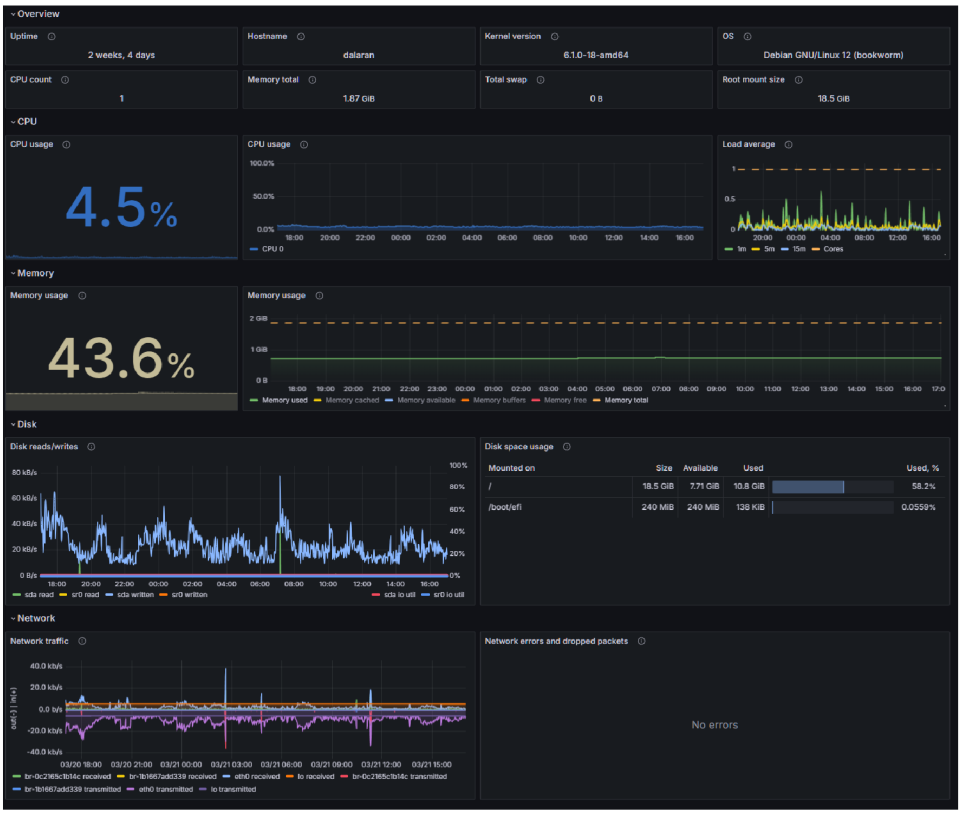
Pokud je potřeba provést jakékoliv změny v hlavním Node.js API, nebo zálohovacím skriptu, tak stačí změny nahrát na hlavní větev daného repositáře. Jakékoliv změny automaticky spustí akci, která vytvoří nový kontejner a nahraje ho na GHCR. Změny hlavního Docker Compose souboru lze na server stáhnout příkazem *git pull*. Nová verze obrazu kontejneru lze stáhnout na server příkazem *docker compose pull* a příkaz *docker compose up -d* aktualizuje současně běžící kontejnery na nejnovější verzi.

3.2.9 Monitoring serveru a kontejnerů

Pro monitoring serveru a všech kontejnerů byla vybrána služba Grafana Cloud. Grafana je široce používaná open-source platforma pro interaktivní vizualizaci dat a sledování metrik. Grafana poskytuje uživatelům grafy, upozornění na náhlé změny v datech, monitoring infrastruktury IT a IoT zařízení a mnoho dalších funkcí. Grafanu lze provozovat jako self-hosted, na vlastním serveru nebo cloudu. Tato možnost je vhodná hlavně pro společnosti, které preferují mít plnou kontrolu nad svými daty. Kromě self-hosted varianty je Grafana také k dispozici jako služba v cloudu. Grafana Cloud je plně spravovaná instance Grafany vhodná pro uživatele, kteří nechtějí řešit provoz a údržbu infrastruktury Grafany samotné a chtějí se zaměřit spíše na využívání dat a jejich analýzu. Grafana Cloud nabízí velmi štědrou bezplatnou verzi disponující všemi funkcemi a je omezená pouze intenzitou používání, počtem sledovaných metrik a úložným prostorem. Zdroje bezplatné verze jsou však pro tuto práci více než dostatečné.

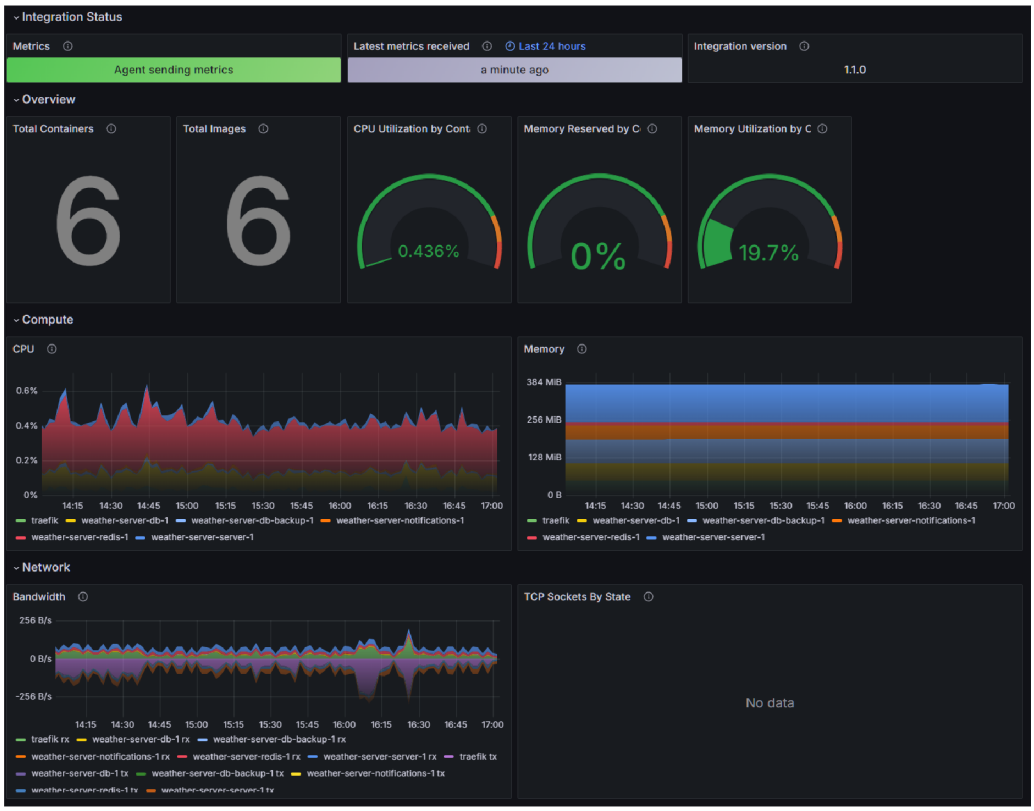
Vytvoření Grafana Cloud instance je velmi jednoduché a po registraci nového bezplatného účtu je potřeba pouze pár kliknutí. Po nastavení nové Grafana instance lze přidávat Integrations, což jsou ve své podstatě doplňky pro Grafanu umožňující snadnou integraci dalších nástrojů, jako jsou např. exportéry dat (Node Exporter), scrappery (Prometheus, Grafana Agent), databáze časových řad (InfluxDB), logovací nástroje (Loki, Elasticsearch) a mnoho dalších nástrojů.

Do Grafany byla nejprve přidána integrace Linux Server sloužící pro monitoring samotného serveru. Veškeré kroky instalace jsou detailně popsány v oficiální dokumentaci Grafany a přidání této integrace zabere pouze pár minut. Integrace využívá Grafana Agent pro shromažďování metrik běžícího operačního systému, včetně využití procesoru, využití paměti a diskové a síťové výstupy / vstupy. Součástí této integrace je 7 předpřipravených dashboardů zobrazující všechny zaznamenané informace.



Obr. 29 Grafana dashboard pro monitoring serveru

Do Grafany byla dále přidána integrace Docker pro sbírání metrik a logů běžícího Docker démona. Tato integrace automaticky detekuje nové Docker kontejnery a sbírá metriky kontejnerů pomocí vestavěného cAdvisoru v Grafana agentovi. Součástí této integrace jsou 2 předpřipravené dashboardy zobrazující všechny zaznamenané informace.



Obr. 30 Grafana dashboard pro monitoring kontejnerů

Součástí těchto dvou integrací jsou i pravidla pro Grafana Alerting, která sledují zdroje dat a vyhodnocují, zda nedošlo k porušení definovaných pravidel. Při porušení těchto pravidel mohou být rozeslána upozornění na různá kontaktní místa jako je např. e-mail, Slack, PagerDuty a další. Tyto pravidla mohou detekovat např. zda se předpokládá, že v blízké dojde systému místo na disku, velmi vysoké využití procesoru po delší dobu a mnoho dalších podmínek.

3.3 Klient

Posledním úkolem této práce bylo vytvořit webovou aplikaci pro vizualizaci dat zaznamenaných meteostanicí. Tato aplikace byla vytvořena v jazycích HTML, CSS a JavaScript (TypeScript). Z důvodů zmíněných v teoretické části této práce byl použit framework SvelteKit s adaptérem *@sveltejs/adapter-static*. Tento adaptér v kombinaci s nastavením *ssr* = *false* a *prerender* = *true* umožňuje předrenderovat celý web jako kolekci statických souborů. Je tak možné velmi jednoduše vytvořit jednostránkovou aplikaci (SPA), která si díky SvelteKitu zachovává routování přes URL adresy.

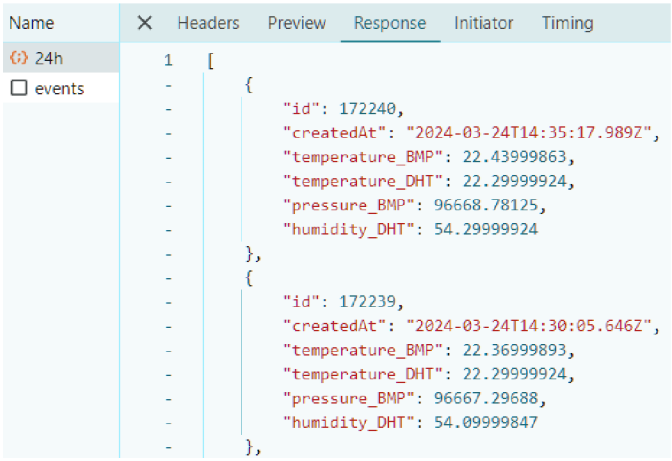
Pro tvorbu klienta byl dále také použit CSS framework Tailwind, který značně usnadňuje psaní CSS. Díky svému utility-first přístupu není potřeba psát rozsáhlé CSS třídy a místo toho lze použít menší předdefinované utility třídy Tailwindu. Tailwind také nabízí předdefinované třídy podporující responzivní design pro různá zařízení a velikosti obrazovky.

Pro kompilaci Svelte kódu byl použit bundler Vite, který tým vývojářů Svelte doporučuje nejvíce. Vite je moderní rychlý nástroj pro vývoj webových aplikací sloužící jako bundler a vývojový server. Vite podporuje hot module replacement (HMR) – při každé změně zdrojového kódu Vite automaticky přebuduje pouze části aplikace, které byly změněny, díky čemuž mají vývojáři velmi rychlou zpětnou vazbu během psaní kódu. Vite má podobně jako Svelte v mnoha oblastech vlastní pevné názory a nabízí automatickou konfiguraci, která většině vývojářů vyhovuje. Zároveň Vite také umožňuje důsledné nastavení Rollupu a esbuildu, které interně využívá pro kompilaci a bundling.

V následujících podkapitolách jsou podrobněji popsány hlavní funkce klienta meteostanice a také jeho nasazení, aby byl přístupný z webového prohlížeče.

3.3.1 Zobrazení zaznamenaných dat

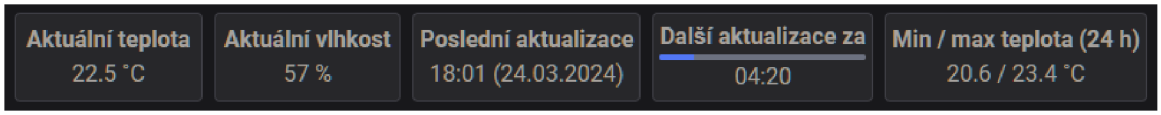
Hlavní funkcí klienta je zobrazovat uživateli zaznamenané hodnoty z meteostanice. K tomuto účelu je nejprve potřeba získat data ze serveru meteostanice na endpointu */readings*. Zde lze získat buď všechna data zaznamenaná za posledních 24 hodin na endpointu */readings/24h*, anebo všechna data zaznamenaná v určitém měsíci na endpointu */readings/month*. Data se získávají posíláním klasických GET požadavků přes standardní JavaScriptové *fetch* API. Odpovědí je vždy JSON pole obsahující zaznamenané hodnoty v daném časovém období. S takto získanými daty lze v klientovi velmi jednoduše pracovat.



Obr. 31 Část odpovědi na GET požadavek na endpoint */readings/24h*

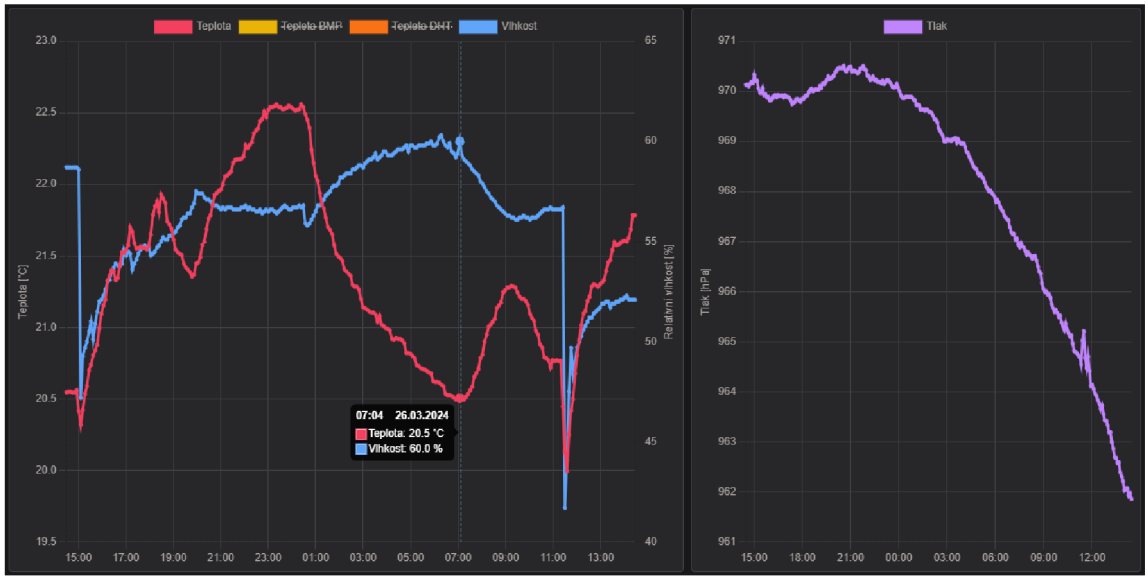
Při každém načtení komponenty hlavní stránky je vždy do Svelte storu uložena nejnovější verze dat zaznamenaných za posledních 24 hodin. Store je jednou z klíčových funkcí Svelte pro správu stavu a je podobný reaktivním proměnným v jiných frameworcích jako je React nebo Vue. Store může ukládat reaktivní stav, ke kterému se mohou připojit jiné komponenty. Když se stav ve store změní, tak se všechny přihlášené komponenty automaticky aktualizují, aby odpovídaly novému stavu. Svelte store s daty za posledních 24 hodin je možné použít kdekoliv v aplikaci.

Z těchto dat jsou vykresleny menší komponenty zobrazující aktuální hodnoty senzorů, minimální a maximální hodnoty, čas posledního měření a předpokládaný čas dalšího měření vypočítaný z průměrné prodlevy mezi předchozími měření.



Obr. 32 Komponenty pro aktuální stav meteostanice

Ze storu s daty za posledních 24 hodin jsou dále vykreslovány interaktivní spojnicové grafy pomocí knihovny Chart.js umožňující rychlou vizualizaci naměřených dat a jejich vývoj v čase.



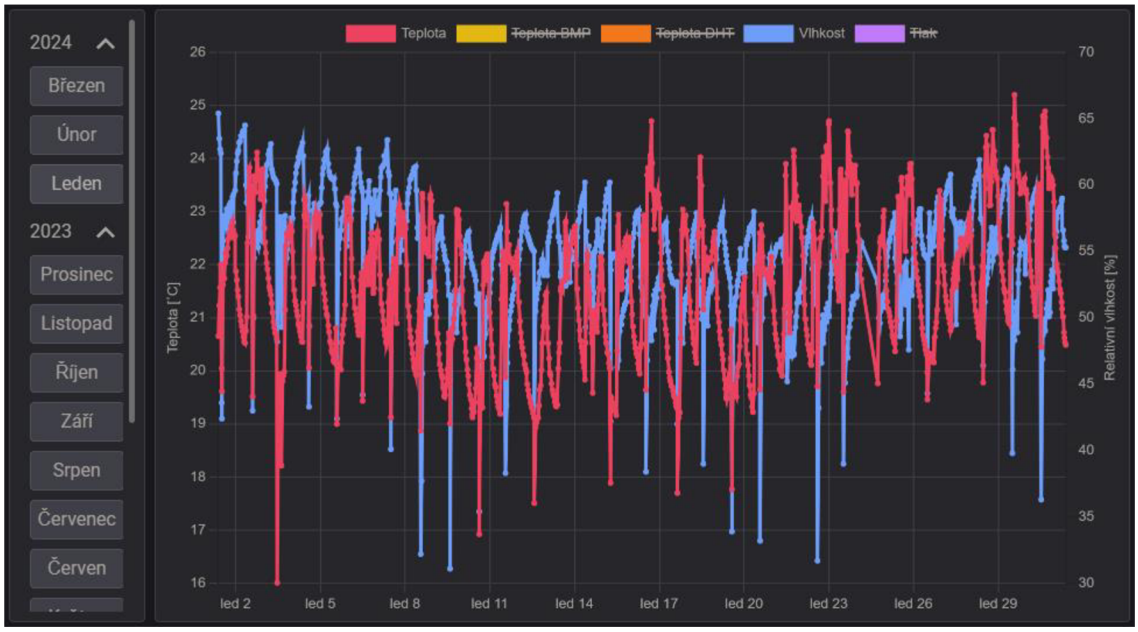
Obr. 33 Grafy se zaznamenanými hodnotami za posledních 24 hodin

Při každém načtení hlavní stránky je vždy automaticky otevřeno SSE (Serve-Sent Events) spojení, pomocí kterého jsou klientovi v reálném čase ze serveru zasílány nově naměřené hodnoty. Nová měření jsou ukládána do storu s hodnotami za posledních 24 hodin. Jak již bylo zmíněno, store je jedním z hlavních prvků reaktivity ve Svelte a všechny změny způsobí překreslení komponent, kde je store použit. Na grafu a ve všech dalších komponentách jsou tak vždy zobrazeny aktuální hodnoty.

```
$: if (!isSSEOpen && isOnline) {
    eventSource = new EventSource(SSE_URL);

    eventSource.addEventListener("message", (event) => {
        const newReading: IReading = JSON.parse(event.data);
        newReading.createdAt = new Date(newReading.createdAt);

        readings.pop();
        readings.unshift(newReading);

        readings = readings;
    });

    eventSource.addEventListener("error", (error) => {
        eventSource.close();

        console.warn("EventSource error:", error);

        setTimeout(() => {
            listeningToSSE = false;
        }, 10_000);
    });

    listeningToSSE = true;
}
```

Zdrojový kód 28 Udržování aktivního spojení a ukládání nových záznamů měření

SSE spojení je udržováno aktivní na straně serveru, ale i straně klienta je nutné kontrolovat, zda nedošlo k nechtěnému ukončení spojení. V takovémto případě je potřeba spojení znovu navázat (např. při dočasné ztrátě internetového připojení).



Obr. 34 Graf s historickými daty

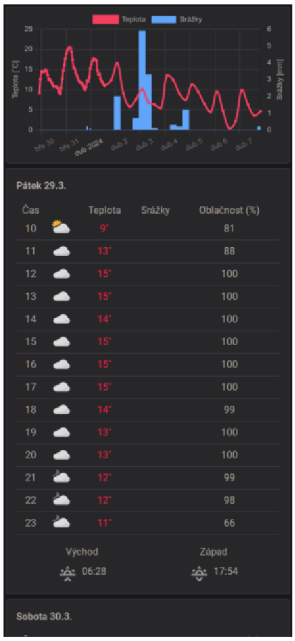
Veškeré záznamy z meteostanice lze také zobrazit na grafu s historickými daty. Zde je možné si zobrazit data z určitého měsíce, anebo také jenom určitý časový úsek dat. Data získaná dotazy na server jsou ukládána nejprve do storu, ze kterého jsou pak vykreslovány podobně jako je tomu u dat za posledních 24 hodin. Všechny dotazy na server jsou ukládány do mezipaměti na straně serveru i v prohlížeči pro rychlejší přístup a minimalizaci síťového provozu mezi serverem a databází a mezi serverem a klientem. Tato část aplikace po stažení dat plně funguje i bez internetového připojení.

3.3.2 Předpověď

Aplikace umožňuje zobrazení předpovědi získané ze serveru. Postup získání předpovědi na serveru je blíže popsán v kapitole *3.2.6*. Prohlížeč získává předpověď ze serveru standardním GET požadavkem na endpoint */forecast*. Odpovědí je JSON obsahující časy východu a západu slunce a pole s předpovědí počasí obsahující hodnoty meteorologických veličin v určitých časových okamžicích.

Obr. 35 Část odpovědi na GET požadavek pro předpověď

Podobně jako u dat z meteostanice je i z předpovědi vykreslován interaktivní spojnicový a sloupcový graf pomocí knihovny Chart.js. Z předpovědi je také vytvořena tabulka zobrazující hodinovou předpověď pro následujících 9 dní společně s časy východu a západu slunce.



Obr. 36 Graf a tabulka s předpovědí počasí

Stejně jako ve zbytku aplikace jsou i požadavky na předpověď ze serveru ukládány do mezipaměti prohlížeče. Hlavička požadavku je nastavena tak, aby jeho platnost vypršela několik minut po aktualizaci modelu předpovědi. V klientovi je tak vždy nejaktuálnější verze předpovědi. V případě, že není k dispozici internetové připojení, je použita poslední uložená verze předpovědi, která je uložena v prohlížeči.

3.3.3 Progresivní webová aplikace

Během vývoje klienta byl kladen důraz na to, aby výslednou aplikaci bylo možné nainstalovat jako progresivní webovou aplikaci (PWA). Pro povolení instalace webové aplikace internetovým prohlížečem musí webová aplikace splňovat určité minimální technické požadavky.

Prvním z požadavků pro klasifikaci webové aplikace jako PWA je service worker obsahující handler *fetch* událostí pro zajištění alespoň základního offline rozhraní a funkcí. Service worker je skript běžící v samostatném kontextu prohlížeče mimo hlavní vlákno a umožňuje aplikaci fungovat offline, ukládat do mezipaměti statické soubory, zachytit a ovládat veškeré síťové požadavky, přijímat push notifikace a mnoho dalšího. Do aplikace byl přidán service worker pomocí knihovny *vite-plugin-pwa*. Knihovna *vite-plugin-pwa* interně využívá Google knihoven *Workbox* pro předběžné ukládání souborů do mezipaměti, které bylo zapnuto pro všechny statické soubory finální zkompilované aplikace. Při prvním navštívení stránky je service worker automaticky registrován a po své instalaci service worker stáhne všechny statické soubory a uloží je do mezipaměti prohlížeče. Při dalších navštívení stránky už service worker pouze kontroluje, zda není k dispozici novější verze souborů. Výsledkem je, že všechny potřebné soubory aplikace jsou uloženy v prohlížeči a při otevření stránky není potřeba stahovat žádné další soubory. Toto značně zrychluje načítání aplikace a celá aplikace funguje i bez připojení k internetu, jelikož service worker má možnost zachytit síťové požadavky a odpovědět na ně například právě daty uloženými v mezipaměti.

Možnost service workera zachytávat a modifikovat síťové požadavky představuje velkou zodpovědnost. Proto je jedním z dalších bezpečnostních požadavků pro instalaci PWA nutnost, aby service worker byl registrován pouze na stránkách používající protokol HTTPS. V současné době většina poskytovatelů webhostingu standardně používá HTTPS, takže registrace service workera je většinou bezproblémová.

Jedním z posledních požadavků pro instalaci PWA je soubor *manifest.json*. Tento soubor určuje, jak se má aplikace v zařízení zobrazovat a chovat.

```
export const manifest: Partial<ManifestOptions> = {
    name: "Meteostanice",
    short_name: "MS",
    description: "Webová aplikace pro sledování dat z meteostanice",
    screenshots: [
        {
            src: "screenshots/narrow/dashboard.png",
            sizes: "420x730",
            form_factor: "narrow",
            type: "image/png",
        },
        ...
    ],
    icons: [
        {
            src: "images/android-chrome-192x192.png",
            sizes: "192x192",
            type: "image/png",
        },
        ...
    ],
    start_url: "/",
    scope: "/",
    display: "standalone",
    orientation: "any",
    theme_color: "#18181b",
    background_color: "#18181b",
};
```

Zdrojový kód 29 Manifest webové aplikace

Zde jsou popsány některé z použitých nastavení manifestu:

- *name, short_name* – viditelný název nainstalované aplikace
- *screenshots* – obrázky určené k prezentaci aplikace před instalací
- *icons* – ikony nainstalované aplikace
- *display* – jak velká část uživatelského rozhraní prohlížeče se uživateli zobrazí
- *orientation* – výchozí orientace aplikace např. na mobilních zařízení
- *theme_color, background_color* – barvy, které má operační systém použít např. pro načítání aplikace, přepínání mezi aplikacemi atd.

Obr. 37 Instalace PWA na mobilním zařízení

PWA s funkčním service workerem a správně vyplněným manifestem lze jednoduše nainstalovat jediným kliknutím po navštívení daných webových stránek. Aplikace má vlastní ikonu na domovské obrazovce mobilního zařízení a lze ji samostatně spustit jako kteroukoliv jinou mobilní aplikaci. Při správně nastaveném manifestu nemá aplikace adresový řádek a ani žádné další prvky prohlížeče a běžný uživatel po instalaci nepozná rozdíl mezi PWA a klasickou nativní aplikací. Postup instalace a použití aplikace na počítači je identický.

3.3.4 Notifikace

Aplikace umožňuje zapnutí přijímání Push notifikací ze serveru. Push zprávy a notifikace jsou dvě samostatné, ale vzájemně se doplňující technologie. Push zprávy slouží pro zasílání libovolných zpráv ze serveru uživatelům, i když aktuálně nepoužívají příslušné webové stránky. Notifikace slouží pro zobrazení informací na zařízení uživatele. Notifikace lze také používat samostatně i bez Push zpráv, ale v praxi se většinou používají společně. Implementace Push notifikací se skládá ze tří hlavních kroků, které jsou popsány v následujících podkapitolách [110].

3.3.4.1 Získání povolení a odběru klienta

Obr. 38 Postup získání povolení a odběru klienta [110]

Prvním krokem pro zapnutí Push notifikací je získání povolení uživatele k zasílání Push notifikací. Např. kliknutím uživatele na tlačítko na stránce je zavolána metoda *Notification.requestPermission()*, která způsobí, že operační systém nebo prohlížeč zobrazí uživateli dialog pro potvrzení souhlasu o zasílání oznámení.

```
try {
    if (Notification.permission === "default") {
       await Notification.requestPermission();
    }

    const permission = Notification.permission;
    if (permission === "denied" || permission === "default") {
        return;
    }

    await createPushSubscription();

    toast.success("Notifikace byly zapnuty", toastOpts.ok);
} catch (error) {
    console.error("Failed to register push notifications", error);
    toast.error("Nepodařilo se zapnout notifikace", toastOpts.err);
}
```

Zdrojový kód 30 Získání povolení uživatele pro zasílání Push notifikací

Po získání povolení lze zahájit proces přihlášení uživatele k odběru Push zpráv pomocí Push API. Nejprve je potřeba získat veřejný VAPID (Voluntary Application Server Identification) klíč ze serveru meteostanice. Veřejný klíč společně s privátním (soukromým) klíčem slouží pro ověření identity serveru, aby se zabránilo neoprávněným serverům v odesílání Push zpráv. Veřejný klíč i privátní klíč byl vygenerován a uložen do proměnných prostředí serveru. Pro získání veřejného klíče byl vytvořen nový endpoint

/push/vapidPublicKey, na kterém klient pomocí GET požadavku může získat hodnotu veřejného klíče.

```
const SWRegistration = await navigator.serviceWorker.ready;

const pushSubscriptionExists = await SWRegistration.pushManager.getSubscription();
if (pushSubscriptionExists) {
    return;
}

const publicKey = await ForecastAPI.getVapidKey();
if (publicKey === undefined) {
    throw new Error("Public VAPID key is required for push subscription");
}

const options: PushSubscriptionOptionsInit = {
    userVisibleOnly: true,
    applicationServerKey: Utils.urlBase64ToUint8Array(publicKey),
};

const pushSubscription = await SWRegistration.pushManager.subscribe(options);

await ForecastAPI.saveSubscription(pushSubscription);
```

Zdrojový kód 31 Přihlášení k odběru Push notifikací

Po získání veřejného klíče serveru může prohlížeč provést síťový požadavek na webovou Push službu pro přihlášení k odběru Push zpráv. Pokud přihlášení proběhlo úspěšně, tak prohlížeč vrátí objekt *PushSubscription* s detaily přihlášení k odběru Push zpráv. Tento objekt je odeslán na server meteostanice, jako součást POST požadavku na endpoint */push/subscribe*. Zde je objekt *PushSubscription* uložen do databáze a pokud vše proběhlo správně, tak je uživateli ihned odeslána Push notifikace, informující uživatele o zapnutí oznámení.

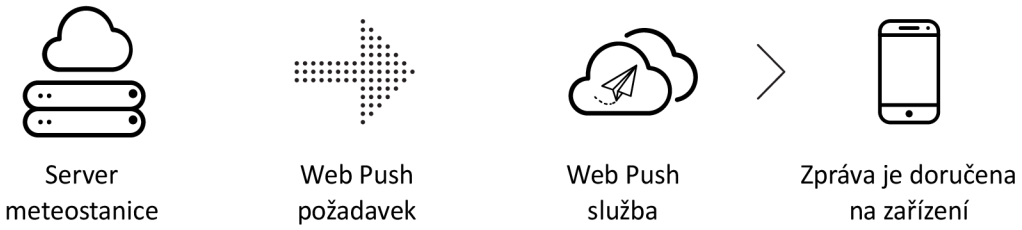
```
public static async saveSubscription(subscription: string) {
    const result = await prisma.pushSubscriptions.create({
        data: {
            pushSubscription: subscription,
        },
    });

    const message = JSON.stringify({
        title: "Meteostanice",
        body: "Notifikace byly zapnuty",
    });

    try {
        await PushService.sendPush(subscription, message);
    } catch (error) {
        await prisma.pushSubscriptions.delete({
            where: { id: result.id },
        });

        throw new AppError(400, "Bad request (invalid subscription)",
        error.message);
    }

    return result;
}
```

Zdrojový kód 32 Služba pro uložení objektu *PushSubscription* do databáze

```
model PushSubscriptions {
  id               Int      @id @default(autoincrement())
  createdAt        DateTime @default(now())
  pushSubscription Json
}
```

Zdrojový kód 33 Prisma model pro uložení *PushSubscription*

3.3.4.2 Odesílání Push zpráv ze serveru

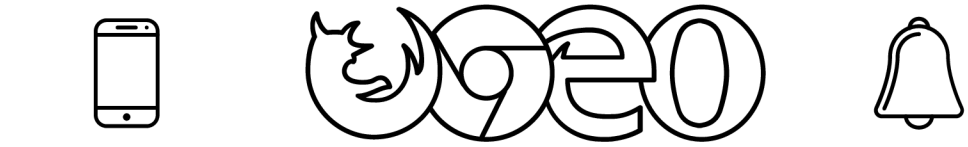


Obr. 39 Postup odeslání a doručení Push zpráv [110]

Push zprávu klientovi neodesílá samotný server meteostanice, ale Push služba. Push služba je webová služba zodpovědná za doručení Push zpráv do zařízení uživatele a je spravována provozovatelem prohlížeče, který uživatel používá. Push služba požadavek ověří a přesměruje Push zprávu příslušnému klientovi. Pokud je prohlížeč klienta offline, Push služba zařadí zprávu do fronty, dokud nebude prohlížeč online nebo dokud nevyprší předem definovaná časová platnost zprávy. Každý prohlížeč používá vlastní Web Push službu a Push požadavky je potřeba poslat na správnou službu, která je definována právě v objektu *PushSubscription*, který prohlížeč vrací po přihlášení uživatele k odběru [110].

Rozesílání Push zpráv ze serveru meteostanice je řízeno knihovnou BullMQ. Tato knihovna mimo vytváření front zpráv, plánování úloh, paralelního zpracování úloh a mnoha dalších funkcí umí také vytvářet úlohy opakující se v určitý čas. Pro odesílání notifikací s předpovědí pro aktuální den byla vytvořena úloha opakující se každý den v šest hodin ráno. Tato úloha nejprve získá kompletní hodinovou předpověď pro daný den, ze které získá hodnoty nejdůležitějších meteorologických veličin a vytvoří z nich krátkou zprávu. Z databáze je získán seznam všech klientů, ze kterého je pro každého klienta na příslušnou Web Push službu zaslán požadavek s vytvořenou zprávou. Pro samotné odesílání Web Push požadavků je používána knihovna web-push.

3.3.4.3 Zobrazení Push notifikací



1. Zpráva je doručena na zařízení

2. Prohlížeč probudí Service Worker

3. Spuštění *push* události

Obr. 40 Postup zobrazení Push notifikací [110]

Po odeslání požadavku na Web Push službu je zpráva uchovávána ve frontě Web Push služby, dokud není klient online a zpráva mu není doručena, nebo dokud nevyprší časová platnost zprávy. Po obdržení Push zprávy prohlížeč zprávu předá service workerovi a spustí *push* událost.

```
self.addEventListener("push", (event) => {
    const data = event.data?.json() ?? {};

    const title = data.title ?? "";
    const body = data.body ?? "";
    const fallbackIcon = "icons/icon-main.svg";
    const icon = data.icon !== undefined ? `icons/weathericons/${data.icon}.svg` :
fallbackIcon;
    const url = "https://weather.bladesheng.com";

    self.registration.showNotification(title, {
        body,
        icon,
        data: {
            url,
        },
    });
});
```

Zdrojový kód 34 Zobrazení notifikace pomocí service workera

Jak již bylo zmíněno, Service worker je skript běžící v samostatném kontextu prohlížeče, který může běžet na pozadí, i když web, anebo prohlížeč není aktuálně otevřený. Service worker naslouchá všem událostem a při spuštění *push* události je zavolána funkce zodpovědná za zobrazení notifikací. Tato funkce z *push* události získá

obsah Push zprávy, která byla odeslána ze serveru, sestaví z ní notifikaci s nadpisem, tělem a ikonou a zobrazí notifikaci uživateli.

Obr. 41 Notifikace s předpovědí počasí

3.3.5 Nasazení klienta

Aby bylo možné aplikaci použít, je nutné aplikaci nejprve zkompilovat ze zdrojového kódu, který je psaný v TypeScriptu a Svelte komponentách do JavaScriptu, kterému prohlížeč rozumí. Jak již bylo zmíněno, pro kompilaci kódu byl použit bundler Vite. Samotná kompilace a sestavení aplikace lze provést přímo po stažení zdrojového kódu na většině operačních systémů. Pro automatizaci a standardizaci kompilace nezávisle na operačním systému byl však vytvořen kontejner zajišťující konzistentní sestavení aplikace.

```
FROM node:20.11.1-bullseye-slim as builder
RUN apt-get update && apt-get upgrade -y
RUN npm -g install pnpm@8.15.5

WORKDIR /app
COPY package.json pnpm-lock.yaml ./
COPY patches patches
ENV NODE_ENV production
RUN pnpm install --frozen-lockfile

COPY . .
RUN pnpm run build
```

Zdrojový kód 35 Kontejner pro sestavení a kompilaci klienta

Základem pro tento kontejner je aktuální verze oficiálního Node obrazu. Do kontejneru jsou staženy a nainstalovány přesné verze všech použitých knihoven, které jsou definovány v souborech *package.json* a *pnpm-lock.yaml*. Zároveň je do kontejneru zkopírována složka *patches* obsahující úpravy zdrojových kódů knihoven, jako je například

oprava orientace popisů grafů knihovny Chart.js. Na závěr je do kontejneru zkopírován zdrojový TypeScript a Svelte kód, který je zkompilován do spustitelného HTML, CSS a JavaScript kódu společně se všemi dalšími statickými soubory, jako jsou obrázky, ikony, fonty a další. Po kompilaci jsou všechny výsledné soubory potřebné pro spuštění aplikace vytvořeny ve složce *build* uvnitř kontejneru.

Pro zobrazení aplikace koncovým uživatelům je potřeba vygenerované soubory umístit na server schopný hostování statických stránek. Pro webhosting byly po předchozích zkušenostech vybrány GitHub Pages, které nabízejí velmi jednoduché bezplatné hostování. Všechen zdrojový kód je uložen v GitHub repositáři, kde lze vytvořit akci pro automatické sestavení a nasazení klienta podobně jako tomu bylo v případě kontejnerů pro server meteostanice. Tato akce má dva hlavní kroky. V prvním kroku akce stáhne zdrojový kód z repositáře do svého běhového prostředí a vytvoří kontejner, který sestaví výsledné statické soubory klienta. Vytvořené soubory jsou z kontejneru přeneseny do běhového prostředí akce, odkud jsou nahrány jako artefakt do dočasného úložiště akce. V druhém kroku akce je artefakt se statickými soubory nasazen na GitHub Pages. Tato akce je automaticky spouštěna vždy při změně kódu v hlavní větvi GitHub repositáře a lze ji také spustit manuálně přes stránky GitHubu. Díky této akci je na GitHub Pages vždy okamžitě nasazena aktuální verze klienta.

```yaml
job-1:
  steps:
    - name: Checkout
      uses: actions/checkout@v4

    - name: Build the Docker image
      run: docker build --file Dockerfile --tag builder-image .

    - name: Run the image to create container
      run: docker run --name builder-container builder-image

    - name: Extract static files from the container
      run: docker cp builder-container:/app/build .

    - name: Upload static files artifact
      uses: actions/upload-pages-artifact@v3
      with:
        path: ./build
        name: github-pages

job-2:
  steps:
    - name: Deploy to GitHub Pages
      id: deployment
      uses: actions/deploy-pages@v4
      with:
        artifact_name: github-pages
```

Zdrojový kód 36 Hlavní kroky akce pro sestavení a nasazení klienta (zkráceno)

4 Závěr

Cílem této diplomové práce bylo vytvořit meteostanici se záznamem dat a online vizualizací časových průběhů environmentálních veličin. Součástí návrhu meteostanice byl výběr vhodného mikrokontroléru a snímačů environmentálních veličin, které nabízí současný trh. Pro tvorbu serveru byly prozkoumány současně dostupné databázové systémy, běhová prostředí serveru, frameworky a použití kontejnerizace na straně serveru. Pro tvorbu webové aplikace byly prozkoumány současně používané frontendové JavaScriptové frameworky.

Hardwarová část meteostanice byla realizována pomocí vývojové desky NodeMCU ESP-32S společně se senzorem teploty vzduchu a vlhkosti vzduchu DHT22 a senzorem atmosférického tlaku a teploty vzduchu BMP280. Mikrokontrolér vývojové desky byl naprogramován v jazyce C++ pro zaznamenávání naměřených hodnot ze senzorů. Zaznamenané hodnoty jsou pravidelně odesílány na server meteostanice přes Wi-Fi rozhraní. Na pronajatém VPS byl pro meteostanici vytvořen server s reverzní proxy umožňující hostování více webových služeb zároveň. Na serveru bylo vytvořeno CRUD API v Node.js a Express.js pro ukládání zaznamenaných dat do Postgres databáze. Všechny nově zaznamenané hodnoty jsou v reálném časem odesílány všem klientům s aktivním SSE spojením. API také umožňuje přihlášení k odběru Web Push notifikací. Dále API meteostanice umožňuje získání předpovědi počasí z externího API. Na server byl přidán Redis pro zrychlení odpovědí na dotazy vyžadující větší objemy dat z databáze. Pro pravidelné zálohování databáze byl vytvořen Python skript, který automaticky ukládá zálohu databáze na externí úložiště, jako je např. Google Cloud. Všechny části API jsou kontejnerizované pomocí Dockeru a lze je tak provozovat v jakémkoliv prostředí nezávisle na konkrétním poskytovateli serverů. Pro monitoring stavu kontejnerů a serveru samotného byl použit Grafana Agent exportující data do Grafana Cloud pro přehledné zobrazení. Webová aplikace pro zobrazení zaznamenaných dat a předpovědi byla vytvořena ve frameworku SvelteKit jako PWA, která funguje na všech platformách s webovým prohlížečem. Push notifikace ze serveru jsou uživateli zobrazovány pomocí service workera. Podobně jako u serverové části bylo sestavení webové aplikace a nasazení automatizováno pomocí GitHub Actions.

Vytvořená meteostanice umožňuje do budoucna snadné rozšíření o další senzory či úpravu funkcionality díky modulární architektuře. Je možné kompletně přepsat jednotlivé části v jiném jazyce či frameworku, což bylo během vývoje několikrát úspěšně učiněno. Plynulost vývoje je do budoucna dále také zajištěna díky použití moderních frameworků a nástrojů ve všech částech meteostanice a také díky vysoce automatizovaném nasazení všech částí aplikace.

Seznam použité literatury

[1] WS View. Online. In: *Google Play*. Dostupné z: https://play.google.com/store/apps/details?id=com.ost.wsview. [citováno 2024-02-05].

[2] GARNI technology. Online. In: *Google Play*. Dostupné z: https://play.google.com/store/apps/details?id=com.garnitechnology.android. [citováno 2024-02-05].

[3] Aplikace. Online. In: *GARNI technology®*. Dostupné z: https://www.garnitechnology.cz/aplikace/. [citováno 2024-02-05].

[4] WeatherLink Cloud. Online. In: *Davis Instruments*. Dostupné z: https://www.davisinstruments.com/pages/weatherlink-cloud. [citováno 2024-02-06].

[5] WeatherLink. Online. In: *Google Play*. Dostupné z: https://play.google.com/store/apps/details?id=com.davisinstruments.weatherlink. [citováno 2024-02-05].

[6] LEPAGE, Pete a Sam RICHARD. What are Progressive Web Apps? Online. In: *web.dev*. Dostupné z: https://web.dev/articles/what-are-pwas. [citováno 2024-02-03].

[7] AL-SHAMMA, Nabeel a Thomas NATTESTAD. Photoshop's journey to the web. Online. In: *web.dev*. Dostupné z: https://web.dev/articles/ps-on-the-web. [citováno 2024-02-03].

[8] *Fugu API Tracker*. Webové sídlo. Dostupné z: https://fugu-tracker.web.app/. [citováno 2024-02-03].

[9] Introduction to client-side frameworks. Online. 2023. In: *MDN Web Docs*. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction. [citováno 2024-01-27].

[10] *State of JavaScript 2022*. Webové sídlo. Dostupné z: https://2022.stateofjs.com. [citováno 2023-12-26].

[11] *React*. Webové sídlo. Dostupné z: https://react.dev/. [citováno 2024-01-27].

[12] HARRIS, Rich. Virtual DOM is pure overhead. Online. 2018. In: *Svelte*. Dostupné z: https://svelte.dev/blog/virtual-dom-is-pure-overhead. [citováno 2024-01-27].

[13] BAI, Aiden. Virtual DOM: Back in Block. Online. 2024. In: *Million.js*. Dostupné z: https://million.dev/blog/virtual-dom. [citováno 2024-01-27].

[14] React.js overview. Online. 2024. In: *Sanity.io*. Dostupné z: https://www.sanity.io/glossary/react-js. [citováno 2024-01-27].

[15] React Labs: What We've Been Working On – March 2023. Online. In: *React*. Dostupné z: https://react.dev/blog/2023/03/22/react-labs-what-we-have-been-working-on-march-2023. [citováno 2024-01-27].

[16] Understanding React Server Components. Online. 2023. In: *Vercel*. Dostupné z: https://vercel.com/blog/understanding-react-server-components. [citováno 2024-01-27].

[17] Vue.js overview. Online. 2024. In: *Sanity.io*. Dostupné z: https://www.sanity.io/glossary/vue-js. [citováno 2024-01-28].

[18] Vue.js guide. Online. In: *Vue.js*. Dostupné z: https://vuejs.org/guide. [citováno 2024-01-28].

[19] HUBSPOT STAFF. What Is Vue.js? Online. 2022. In: *HubSpot*. Dostupné z: https://blog.hubspot.com/website/vue-js. [citováno 2024-01-28].

[20] Getting started with Svelte. Online. 2023. In: *MDN Web Docs*. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Svelte_getting_started. [citováno 2024-01-29].

[21] HARRIS, Rich. Svelte 3: Rethinking reactivity. Online. 2019. In: *Svelte*. Dostupné z: https://svelte.dev/blog/svelte-3-rethinking-reactivity. [citováno 2024-01-29].

[22] Docs. Online. In: *Svelte*. Dostupné z: https://svelte.dev/docs. [citováno 2024-01-29].

[23] THE SVELTE TEAM. Introducing runes. Online. 2023. In: *Svelte*. Dostupné z: https://svelte.dev/blog/runes. [citováno 2024-01-29].

[24] Sveltekit overview. Online. 2024. In: *Sanity.io*. Dostupné z: https://www.sanity.io/glossary/sveltekit. [citováno 2024-01-29].

[25] Docs. Online. In: *SvelteKit*. Dostupné z: https://kit.svelte.dev/docs. [citováno 2024-01-29].

[26] Stack Overflow Developer Survey 2022. Online. In: *Stack Overflow*. Dostupné z: https://survey.stackoverflow.co/2022/. [citováno 2023-11-18].

[27] Stack Overflow Developer Survey 2023. Online. In: *Stack Overflow*. Dostupné z: https://survey.stackoverflow.co/2023/. [citováno 2023-11-18].

[28] AHINON, Justin. Top 10 Big Companies Using Svelte. Online. 2023. In: *Okupter*. Dostupné z: https://www.okupter.com/blog/companies-using-svelte. [citováno 2024-01-29].

[29] JavaScript. Online. 2023. In: *MDN Web Docs*. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [citováno 2023-12-27].

[30] Ecma TC39. Online. In: *GitHub*. Dostupné z: https://github.com/tc39. [citováno 2023-12-27].

[31] TC39. Online. In: *Ecma International*. Dostupné z: https://ecma-international.org/technical-committees/tc39/. [citováno 2023-12-27].

[32] JavaScript technologies overview - JavaScript | MDN. Online. 2023. In: *MDN Web Docs*. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/JavaScript_technologies_overview. [citováno 2023-12-27].

[33] The TC39 Process. Online. In: *TC39*. Dostupné z: https://tc39.es/process-document/. [citováno 2023-12-27].

[34] ECMAScript proposals. Online. In: *GitHub*. Dostupné z: https://github.com/tc39/proposals/blob/main/README.md. [citováno 2023-12-27].

[35] TC39 Royalty Free Technical Committee members. Online. In: *Ecma International*. Dostupné z: https://ecma-international.org/tc39-royalty-free-technical-committee-members/. [citováno 2023-12-27].

[36] BERNHARDT, Gary (režie). *Wat*. online. 2012. Dostupné z: https://www.destroyallsoftware.com/talks/wat.

[37] FIRESHIP (režie). *JavaScript for the Haters*. online. 2022. Dostupné z: https://www.youtube.com/watch?v=aXOChLn5ZdQ.

[38] ECMAScript proposal: Type Annotations. Online. 2023. In: *GitHub*. Dostupné z: https://github.com/tc39/proposal-type-annotations. [citováno 2023-12-27].

[39] 27 September, 2023 TC39 Meeting Notes. Online. In: *GitHub*. Dostupné z: https://github.com/tc39/notes/blob/main/meetings/2023-09/september-27.md. [citováno 2023-12-27].

[40] *TypeScript*. Webové sídlo. Dostupné z: https://www.typescriptlang.org/. [citováno 2023-12-28].

[41] *Node.js*. Webové sídlo. Dostupné z: https://nodejs.org/en. [citováno 2023-12-02].

[42] Introduction to npm. Online. In: *Node.js*. Dostupné z: https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager. [citováno 2023-12-02].

[43] How many npm users are there? Online. In: *npm Blog*. Dostupné z: https://blog.npmjs.org/post/143451680695/how-many-npm-users-are-there.html. [citováno 2023-12-02].

[44] *npm*. Webové sídlo. Dostupné z: https://www.npmjs.com/. [citováno 2023-12-02].

[45] DAHL, Ryan (režie). *10 Things I Regret About Node.js - Ryan Dahl - JSConf EU*. online. 2018. Dostupné z: https://www.youtube.com/watch?v=M3BM9TB-8yA.

[46] *Deno*. Webové sídlo. Dostupné z: https://deno.com/. [citováno 2023-12-07].

[47] Permissions. Online. In: *Deno Docs*. Dostupné z: https://docs.deno.com/runtime/manual/basics/permissions. [citováno 2023-12-07].

[48] Suggestion: Look into porting to Rust and using Tokio. Online. In: *GitHub*. Dostupné z: https://github.com/denoland/deno/issues/205. [citováno 2023-12-07].

[49] What is Bun? Online. In: *Bun*. Dostupné z: https://bun.sh/docs. [citováno 2023-12-09].

[50] GOPINATH, Vishwas. Bun vs Node.js: Everything you need to know. Online. In: *Builder.io*. Dostupné z: https://www.builder.io/blog/bun-vs-node-js. [citováno 2023-12-10].

[51] SUMMER, Jared. Bun 1.0. Online. 2023. In: *Bun*. Dostupné z: https://bun.sh/blog/bun-v1.0. [citováno 2023-12-09].

[52] AHMOD, Md Feroj. Javascript runtime performance analysis: Node and Bun. online. 2023. Dostupné z: https://trepo.tuni.fi/handle/10024/149672.

[53] *Bun*. Webové sídlo. Dostupné z: https://bun.sh. [citováno 2023-12-10].

[54] Bun's priorities. Online. In: *GitHub*. Dostupné z: https://github.com/oven-sh/bun/issues/798. [citováno 2023-12-09].

[55] Issues - Bun. Online. In: *GitHub*. Dostupné z: https://github.com/oven-sh/bun/issues. [citováno 2023-12-09].

[56] Issues - Node. Online. In: *GitHub*. Dostupné z: https://github.com/nodejs/node/issues. [citováno 2023-12-09].

[57] Issues - Deno. Online. In: *GitHub*. Dostupné z: https://github.com/denoland/deno/issues. [citováno 2023-12-09].

[58] *NestJS Docs*. Webové sídlo. Dostupné z: https://docs.nestjs.com. [citováno 2024-01-08].

[59] Express/Node introduction. Online. 2024. In: *MDN Web Docs*. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction. [citováno 2024-01-08].

[60] *Sails.js*. Webové sídlo. 2024. Dostupné z: https://sailsjs.com/. [citováno 2024-01-08].

[61] Express history. Online. In: *GitHub*. Dostupné z: https://github.com/expressjs/express/blob/master/History.md. [citováno 2024-01-08].

[62] Express middleware. Online. In: *Express*. Dostupné z: https://expressjs.com/en/resources/middleware.html. [citováno 2024-01-08].

[63] *Fastify*. Webové sídlo. Dostupné z: https://fastify.io/. [citováno 2024-01-20].

[64] Fastify plugins. Online. In: *Fastify*. Dostupné z: https://fastify.io/ecosystem/. [citováno 2024-01-20].

[65] What Are Databases? Online. In: *Microsoft Azure*. Dostupné z: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-are-databases. [citováno 2023-11-30].

[66] *What is a database?*. Webové sídlo. Dostupné z: https://www.oracle.com/database/what-is-database/. [citováno 2023-11-30].

[67] Database management systems. Online. 2023. In: *IBM Documentation*. Dostupné z: https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-what-is-database-management-system. [citováno 2023-11-30].

[68] *MongoDB*. Webové sídlo. Dostupné z: https://www.mongodb.com. [citováno 2023-11-19].

[69] VAN DER VEEN, Jan Sipke; Bram VAN DER WAAIJ a Robert J. MEIJER. Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual. 2012-06. s. 431–438.

[70] *PostgreSQL*. Webové sídlo. 2023. Dostupné z: https://www.postgresql.org/. [citováno 2023-11-18].

[71] SALVA, Pavel. Analýza a optimalizace indexů v PostgreSQL. 2021.

[72] Usage Statistics and Market Share of Content Management Systems, November 2023. Online. In: *W3Techs*. Dostupné z: https://w3techs.com/technologies/overview/content_management. [citováno 2023-11-18].

[73] *MySQL*. Webové sídlo. Dostupné z: https://www.mysql.com/. [citováno 2023-11-18].

[74] JOHNSON, Renee a Alberto MEDINA. *The 2019 Web Almanac: CMS*. online. HTTP Archive. 2019. Dostupné z: https://almanac.httparchive.org/en/2019/cms. [citováno 2023-11-18].

[75] LARDINOIS, Frederic. *MongoDB switches up its open-source license*. 2018-10-16. Dostupné z: TechCrunch, https://techcrunch.com/2018/10/16/mongodb-switches-up-its-open-source-license/. [citováno 2023-11-19].

[76] About SQLite. Online. In: *SQLite*. Dostupné z: https://www.sqlite.org/about.html. [citováno 2023-11-23].

[77] Most Widely Deployed SQL Database Engine. Online. In: *SQLite*. Dostupné z: https://www.sqlite.org/mostdeployed.html. [citováno 2023-11-23].

[78] What is Docker? | IBM. Online. In: *IBM*. Dostupné z: https://www.ibm.com/topics/docker. [citováno 2024-01-22].

[79] What is Docker? Online. In: *Amazon Web Services, Inc.* Dostupné z: https://aws.amazon.com/docker/. [citováno 2024-01-22].

[80] What is a Container? Online. In: *Docker*. Dostupné z: https://www.docker.com/resources/what-container/. [citováno 2024-01-22].

[81] *Docker Documentation*. Webové sídlo. 100n. l. Dostupné z: https://docs.docker.com/. [citováno 2024-01-22].

[82] What is Docker? Online. In: *Oracle*. Dostupné z: https://www.oracle.com/cz/cloud/cloud-native/container-registry/what-is-docker/. [citováno 2024-01-22].

[83] 2023 State of the Cloud. Online. In: *Flexera*. Dostupné z: https://info.flexera.com/CM-REPORT-State-of-the-Cloud. [citováno 2024-01-22].

[84] What Is Kubernetes? Online. In: *Google Cloud*. Dostupné z: https://cloud.google.com/learn/what-is-kubernetes. [citováno 2024-01-23].

[85] STM32MCU basics. Online. In: *STMicroelectronics*. Dostupné z: https://wiki.st.com/stm32mcu/wiki/STM32StepByStep:STM32MCU_basics. [citováno 2024-02-11].

[86] STM32 32-bit Arm Cortex MCUs. Online. In: *STMicroelectronics*. Dostupné z: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html. [citováno 2024-02-11].

[87] STM32 Nucleo Boards. Online. In: *STMicroelectronics*. Dostupné z: https://www.st.com/en/evaluation-tools/stm32-nucleo-boards.html. [citováno 2024-02-12].

[88] NUCLEO-F070RB. Online. In: *STMicroelectronics*. Dostupné z: https://www.st.com/en/evaluation-tools/nucleo-f070rb.html. [citováno 2024-02-12].

[89] Products. Online. In: *Nordic Semiconductor*. Dostupné z: https://www.nordicsemi.com/Products. [citováno 2024-02-12].

[90] HETTING, Claus. *Nordic Semiconductor targets doubling revenues by 2026 aided by new line of Wi-Fi IoT chipsets*. 2022-10-12. Dostupné z: Wi-Fi NOW Global, https://wifinowglobal.com/news-and-blog/nordic-semiconductor-targets-doubling-revenues-by-2026-aided-by-new-line-of-wi-fi-iot-chipsets/. [citováno 2024-02-12].

[91] nRF52 DK. Online. In: *Nordic Semiconductor*. Dostupné z: https://www.nordicsemi.com/Products/Development-hardware/nRF52-DK. [citováno 2024-02-12].

[92] nRF9160 DK. Online. In: *Nordic Semiconductor*. Dostupné z: https://www.nordicsemi.com/Products/Development-hardware/nRF9160-DK. [citováno 2024-02-12].

[93] Features of nRF70 Series. Online. In: *Nordic Semiconductor*. Dostupné z: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/device_guides/working_with_nrf/nrf70/features.html. [citováno 2024-02-12].

[94] Insight Into ESP32 Sleep Modes & Their Power Consumption. Online. 2018. In: *Last Minute Engineers*. Dostupné z: https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/. [citováno 2024-02-13].

[95] *ESP-IDF Docs*. Webové sídlo. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html. [citováno 2024-02-13].

[96] Development Boards. Online. In: *Espressif*. Dostupné z: https://www.espressif.com/en/products/devkits. [citováno 2024-02-13].

[97] *Capacitive-type humidity and temperature module/sensor DHT22*online. Dostupné z: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf.

[98] HWKITCHEN. SNÍMAČ TEPLOTY A VLHKOSTI DHT22. Online. In: *HWKitchen*. Dostupné z: https://www.hwkitchen.cz/snimac-teploty-a-vlhkosti-dht22/. [citováno 2024-02-15].

[99] AOSONG. *AM2301 Product Manua*online. Dostupné z: https://www.haoyuelectronics.com/Attachment/AM2301/AM2301.pdf.

[100] Capacitive Digital Temperature & Humidity Sensor (AM2301). Online. In: *HAOYU Electronics*. Dostupné z: https://www.hotmcu.com/capacitive-digital-temperature-humidity-sensoram2301-p-154.html. [citováno 2024-02-17].

[101] BOSCH SENSORTEC. *BMP280 Datasheet*online. Dostupné z: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001.pdf.

[102] Senzor barometrického tlaku a teploty BMP280. Online. 2021. In: *LaskaKit*. Dostupné z: https://www.laskakit.cz/arduino-senzor-barometrickeho-tlaku-a-teploty-bmp280/. [citováno 2024-02-17].

[103] BOSCH SENSORTEC. *BME680 Datasheet*online. Dostupné z: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme680-ds001.pdf.

[104] BME680 - snímač teploty, vlhkosti, tlaku a plynu I2C / SPI - STEMMA QT / Qwiic - Adafruit. Online. 2030. In: *BOTLAND*. Dostupné z: https://botland.cz/multifunkcni-senzory/10872-bme680-snimac-teploty-vlhkosti-tlaku-a-plynu-i2c-spi-stemma-qt-qwiic-adafruit-3660-5904422316914.html. [citováno 2024-02-17].

[105] SENSIRION. *SHTC3 Datasheet*online. Dostupné z: https://sensirion.com/media/documents/643F9C8E/63A5A436/Datasheet_SHTC3.pdf.

[106] Adafruit Sensirion SHTC3. Online. In: *Mouser Electronics*. Dostupné z: https://eu.mouser.com/new/adafruit/adafruit-shtc3-sensor/. [citováno 2024-02-17].

[107] AI-THINKER. *Nodemcu-32s Datasheet*online. 2019. Dostupné z: https://docs.ai-thinker.com/_media/esp32/docs/nodemcu-32s_product_specification.pdf.

[108] traefik. Online. 2024. In: *GitHub*. Dostupné z: https://github.com/traefik/traefik. [citováno 2024-03-02].

[109] STEINARSSON, Sveinn. *Downsampling Time Series for Visual Representation*. University of Iceland, 2013. Dostupné z: https://skemman.is/bitstream/1946/15343/3/SS_MSthesis.pdf.

[110] BASQUES, Kayce a Matt GAUNT. Push notifications overview. Online. In: *web.dev*. Dostupné z: https://web.dev/articles/push-notifications-overview. [citováno 2024-03-31].

Příloha A – Obsah CD

Na přiloženém CD se nachází:

- RadekTvrznik.dp.pdf – tato práce ve formátu PDF,
- weather-station.zip – zdrojové kódy aplikace.