Systém pro sběr a zpracovávání meteorologických dat

Bc. Michal Horák

Diplomová práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Michal Horák**
Osobní číslo:	**A20130**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Systém pro sběr a zpracovávání meteorologických dat**
Téma práce anglicky:	**System for Collecting and Processing Meteorological Data**

Zásady pro vypracování

1. Popište současný stav technologií pro implementaci a zabezpečení domácí meteostanice, webového klienta a serveru.
2. Zvolte vhodné technologie pro serverovou a klientskou část.
3. Navrhněte aplikaci, definujte funkční a nefunkční požadavky, případy použití a případě další modely.
4. Zvolte vhodný způsob zabezpečení komunikace mezi webovým klientem, serverem a meteorologickou stanicí.
5. Realizujte vývoj navržené aplikace a popište její klíčové části.
6. Demonstrujte výsledky a formulujte závěr.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. BLUM, Jeremy. Exploring Arduino: tools and techniques for engineering wizardry. Second edition. Indianapolis: Wiley, [2020]. ISBN 9781119405375.
2. UPTON, Eben a Gareth HALFACREE. Raspberry Pi: uživatelská příručka. 2., aktualizované vydání. Přeložil Jakub GONER. Brno: Computer Press, 2016. ISBN 978-80-251-4819-8.
3. AMUNDSEN, Mike. RESTful Web Clients. 1. Sebastopol, USA: O'Reilly Media, 2017. ISBN 9781491921906.
4. Dokumentace k ASP.NET [online]. Redmond: Microsoft, 2021 [cit. 2021-9-20]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/?view=aspnetcore-5.0
5. PRICE, Mark J. C# 9 and .NET 5 – Modern Cross-Platform Development. 5. Birmingham: Packt Publishing, 2020. ISBN 9781800568105.
6. WRIGHT, Toi B. Blazor WebAssembly by Example. 1. Birmingham: Packt Publishing, 2021. ISBN 9781800567511.
7. Dokumentace k jazyku C# [online]. Redmond: Microsoft, 2021 [cit. 2021-9-19]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/csharp/

Vedoucí diplomové práce: **Ing. Erik Král, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **3. prosince 2021**
Termín odevzdání diplomové práce: **23. května 2022**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Bc. Michal Horák, v. r.
podpis studenta

ABSTRAKT

Cílem této diplomové práce je vytvořit systém, který bude umožňovat sbírat data z připojených meteostanic a následně získaná data prezentovat uživateli systému.

Teoretická část práce se zaměřuje na popis použitých technologií a některých jejich alternativ, dále na hardware jak pro hostování serveru, tak pro konstrukci meteostanice. Taktéž je zde pojednáváno o základech meteorologie jako takové.

Praktická část pak popisuje motivaci k tvorbě aplikace, návrh systému a definování požadavků, dále vývoj a sestavení systému, a nakonec jeho reálný provoz. Na závěr je zde popsána možná rozšiřitelnost a další zlepšení systému za účelem zvýšení potencionálu celého systému.

Klíčová slova: C#, .NET, ASP.NET, Blazor, REST API, Arduino, Raspberry Pi, Meteorologie

ABSTRACT

Purpose of this diploma thesis is to create a system, which will be able to collect data from connected weather stations and display gathered data to user of this system.

Theoretical part is focused on description of used technologies and some of their alternatives, then on hardware for hosting the application server and for construction of weather station itself. The basics of meteorology as such are also discussed here.

Practical part includes motivation that led to creating of this application, design of system itself and definitions of requirements. This part also includes the process of development, assembly of system and its operation in real conditions. Finally, the possible scalability and further improvement of the system in order to increase the potential of the whole system is described here.

Keywords: C#, .NET, ASP.NET, Blazor, REST API, Arduino, Raspberry Pi, Meteorology

Rád bych poděkoval vedoucímu práce, Ing. et Ing. Erik Král, Ph.D., za rady a pomoc při tvorbě této diplomové práce.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Počasí bylo už od počátků lidstva důležitým faktorem u různých rozhodování. Klima obecně ovlivňuje, tak se lidé oblékají nebo z jakého materiálů a v jaké lokalitě si staví příbytky. Vědět proto, jaké jsou aktuální podmínky a jaké budou za nějakou dobu, je poměrně důležitá informace.

Jednou z možností, jak aktuální počasí zjišťovat, je pomocí automatizovaných zařízení, zvaných meteorologická stanice, či zkráceně meteostanice. Ty pomocí speciálních senzorů pozorují okolní podmínky a tyto data zaznamenávají a zobrazují uživatelům.

Cílem této práce je vytvořit systém, který bude meteorologická data získávat, ukládat, zpracovávat a poté je zobrazí uživateli. Systém samotný se bude skládat ze softwarové části, rozdělené na klientskou část určenou pro zobrazování dat, kontrolu a nastavení aplikace samotné, a serverové části, starající se o vyřizování požadavků od uživatele a o správu veškerých informací v aplikaci, četně autorizace a kontroly dat. Další části systému je hardware, kdy jedna část je určena pro hostování této aplikace, a druhá, ve formě meteostanice samotné, pro získávání meteorologických dat.

Implementace tohoto systému je provedena v programovacím jazyku C# a pomocí platformy .NET, která zahrnuje nástroje určené jak pro tvorbu klientské části ve formě webových stránek, tak i pro serverovou část, zahrnující přístupové koncové body. Co se týká kódu pro meteostanice, ten je psán pomocí jazyku a platformy Wiring, což je zjednodušeně řečeno nadstavba jazyka C++.

I. TEORETICKÁ ČÁST

1 SOFTWARE

Pro tvorbu softwarového systému jsou za potřebí různé druhy softwaru. Patří zde hlavně programovací (a další) jazyky a s nimi spojené frameworky, každý vytvořen k jinému účelu a použití. Dále to jsou integrované vývojové prostředí, určené pro aplikování jazyka k vývoji aplikace a k jejímu následnému sestavení pomocí nástrojů zahrnutých v některých těchto prostředích. Součástí jsou zde i některé frameworky, ulehčující práci při vývoji nebo jinak rozšiřující funkcionalitu výsledného produktu.

1.1 Programovací jazyky

Co se týče programovacích jazyků, existuje jich spousta druhů, lišících se v detailech, ale i v celkové koncepci, v jejich funkcionalitě nebo ve způsobu překladu. Každý z programovacích jazyků je i určen pro jiný typ úloh, jiný rozsah a jiné určení.

Zde uvedené jazyky byly buď přímo, nebo nepřímo použity pro tvorbu samotného systému. Jako nepřímo použitý jazyk je považován například zde uvedený JavaScript, ve kterém bylo vytvořeno pouze pár řádků kódu, ovšem dále uvedený framework Blazor ho v rámci jeho varianty WebAssembly využívá ke svému fungování.

1.1.1 C / C++

1.1.1.1 Jazyk C

Programovací jazyk C je procedurální, kompilovaný programovací jazyk, který byl vyvinut v roce 1972 Denisem Ritchiem a Brianem Kerninghanem jako pomyslná evoluce jazyka B [1]. Jeho účelem byl v té době hlavně vývoj operačních systémů, proto obsahuje řadu „nízkoúrovňových“ funkcí, zaměřené právě k tomuto účelu. Jedná se například o nízko úrovňový přístup do paměti, jednoduchá klíčová slova a čistou syntaxi. Uvedená charakteristika ovšem nemusí být vždy výhodou. Jazyk sice umožňuje tvorbu vysoce efektivního kódu, ale kód samotný se z velké části musí vytvářet tzv. od nuly, protože jazyk nezahrnuje některé pokročilejší funkce nebo struktury či typy. [2] Naštěstí existuje celá řada knihoven, které potřebné funkce poskytují.

V dnešní době je tento jazyk stále populární a používaný. Popularitu lze pozorovat na rozličných statistikách, například v průzkumu Stack Overflow z roku 2021 se v popularitě jazyků umístilo C na 12. místě z 21 % ze všech respondentů [3]. Co se týká použití, tak mimo

původní účel jako jazyk pro psaní operačních systémů je vhodný i pro tvorbu aplikací, kde je hlavním požadavkem výkon, a pro programování embedded a IoT zařízení.

1.1.1.2 Jazyk C++

Jako rozšíření jazyka C vznikl v roce 1985 jazyk C++, taky nazývaný „C s třídami“ [4]. Jak název napovídá, oproti původnímu jazyku přidává možnost objektově orientovaného programování. To zahrnuje koncepty jako jsou objekty, třídy, dědičnost, abstrakce a další. I přes tyto změny si jazyk C++ ponechává hlavní výhody jazyka C, jako je rychlost a efektivita.

Co se týče kompatibility obou jazyků, tak platí, že kód napsaný v C je zároveň kódem v jazyce C++. Opačným směrem to ovšem neplatí, jelikož jazyk C++ přidává vlastní třídy a syntaxi, která v základním C není obsažena. Výhodou této kompatibility je bohatý výběr podporovaných knihoven, ze kterých lze čerpat během vývoje.

Použitím se tento jazyk částečně kryje s jazykem C. Využívá se pro psaní efektivních a výkonných aplikací, jako příklad lze uvést operační systémy, webové prohlížeče nebo i počítačové hry. [4]

1.1.2 C#

Jazyk C# je silně typový, objektově orientovaný programovací jazyk, vytvořený a vyvíjený společností Microsoft. Smyslem jazyka bylo vyvinout programovací jazyk, který je jednoduchý na naučení a zároveň podporuje moderní funkcionality pro vývoj různorodého softwaru. [5]

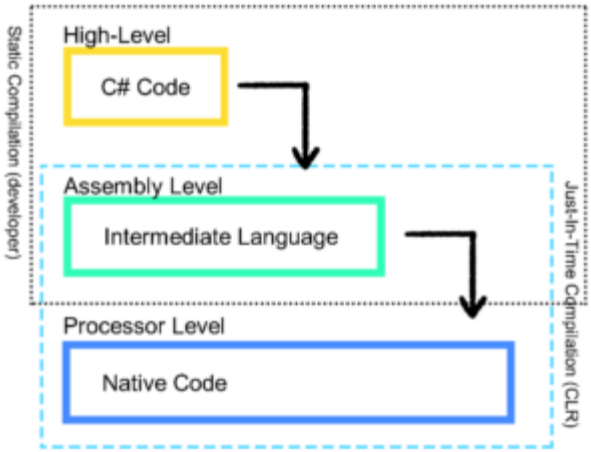
1.1.2.1 Historie

V době psaní této práce nachází už ve své desáté verzi a od jeho vzniku prošel jazyk hodně změnami. Přibyla mu řada užitečných funkcionalit, které mu umožňují být stále konkurenceschopný a atraktivní pro nové i stávající vývojáře. Jako jeden z klíčových okamžiků vývoje lze pravděpodobně považovat přechod na Open Source, čímž bylo umožněno komunitě částečně zasahovat do směru, kterým se bude další vývoj jazyka směřovat a jaké další užitečné funkce jazyk získá. [5]

1.1.2.2 Kompilace

Kompilování jazyka se od většiny ostatních liší. Konkrétně se jedná o kombinaci interpretovaného a kompilovaného jazyka. Jako interpretovanou část lze považovat převod samotného

C# kódu do tzv. Common Intermediate Language. V této formě lze program distribuovat na jiná zařízení, ovšem není možné program přímo spustit. Proto v dalším kroku je nutné ho převést na kód, kterému už bude daný systém rozumět a dovede ho spustit. K tomu je určena tzv. Common Language Runtime, která převede kód z přechodného jazyka na nativní kód. V případě jazyka C# je tento proces součástí frameworku .NET, u jazyka Java lze přirovnat k systému Java Virtual Machine [6].



Obrázek 1 Proces kompilace jazyka C# [6]

1.1.2.3 Rodina jazyků

Samotný jazyk je členem rodiny jazyků C, zahrnující řadu jazyků, který konceptuálně vycházejí z programovacího jazyka C a jeho předchůdců. Ta zahrnuje jak jazyky s písmenem C v názvu, ale i jiné podobné jazyky, jako je například Java nebo PHP.

1.1.2.4 Přednosti

Jednou z výhod jazyka je jeho univerzálnost, což nemusí platit u jiných jazyků, které byly většinou vytvořeny za jedním specifickým cílem. Spolu s platformou .NET s ním lze vytvářet multiplatformní desktopové, mobilní nebo i webové aplikace. Taktéž s ním lze programovat i počítačové hry, zařízení pro Internet věcí nebo ho lze použít na komponenty pro již existující software. [5]

Za další plus lze považovat jeho aktivní vývoj. To zajišťuje neustálý přísun nových a vylepšování stávajících funkcí, opravování chyb a „zalepování“ zneužitelných slabin. Tím jazyk zůstává stále aktuální, bezpečný, a hlavně atraktivní pro nové i stávající vývojáře.

1.1.3 JavaScript

JavaScript je dynamický programovací jazyk, nejčastěji používaný jako součást webových stránek, umožňující dynamicky reagovat na požadavky uživatele [7]. I přes podobný název s programovacím jazykem Java mají oba jazyky pramálo společného.

I když je jazyk primárně zaměřen na tvorbu uživatelských částí webových stránek a aplikací, není to jeho jediné možné uplatnění. Lze ho také využít pro tvorbu mobilních aplikací, klientské i serverové části webových aplikací, a dokonce i pro tvorbu počítačových her, a to jak webových, tak i desktopových.

Výhodami jazyka je omezení nutnosti komunikace se serverem, například při aktualizaci zobrazení u uživatele v reakci na nějakou jeho akci, čímž umožňuje velmi rychlou reakci na případné další požadavky. Taktéž dovoluje vytvářet interaktivní webové stránky, například definováním akcí při provedení nějaké pohybu myší nebo vybráním položky.

Jazyk má také pár nevýhod, často vytvořené úmyslně kvůli bezpečnosti. Jedná se například o přímé zapisování a čtení souborů, kvůli možnému zneužití. Javascriptu taktéž chybí podpora multithreadingu, tedy využití více vláken / jader procesoru. [7]

Z hlediska rozšířenosti a oblíbenosti se dnes jedná o jeden z nejpopulárnějších programovacích jazyků, jak napovídá průzkum mezi komunitou stránky Stack Overflow [3]. Za to vděčí hlavně tomu, že je to de facto standard, co se týče webových technologií, který je dále rozšiřován frameworky, jako je Node.js nebo třeba nadstavbovými jazyky, jako například TypeScript [8].

1.2 Popisovací jazyky

V této části jsou uvedeny jazyky určené k tvorbě webových stránek, pro práci z databází a obecně popisovací jazyky, sloužící k ukládání dat nebo popisování vzhledu souborů a stránek. Jazyky samotné neslouží k tvorbě nějaké funkcionality, ale pouze pro ukládání dat nebo pro jejich vizualizaci.

1.2.1 HTML

Hypertextový značkovací jazyk je v dnešní době společně s kaskádovými styly a JavaScriptem standard na tvorbu vzhledu a funkcionality webových stránek. Nejedná se o programovací, ale o značkovací jazyk, na základě, kterého se vykresluje obsah stránky. Tento popis je založen na skládání elementů, každý s daným tvarem a účelem. Oproti většině programovacích jazyků také dokáže do určité míry tolerovat chyby a webový prohlížeč se pokouší zobrazit obsah stránky za „každou cenu“.

Samotný jazyk lze pokládat za stavební blok webu a webových stránek, který definuje smysl a strukturu webového obsahu. „Hypertext“ v názvu odkazuje na možnost propojovat webové stránky mezi sebou, buďto v rámci jednoho nebo i více různých webů. Tato spojení jsou proto pomyslným základním aspektem webu samotného a umožňují aktivní zapojení do sítě World Wide Web. [9]

Jelikož se jedná o standard, podporují ho všechny prohlížeče a většina vývojových prostředí, i když syntaxe jazyka je poměrně jednoduchá a webová stránka se dá pomocí něho vytvořit i v „hloupém“ poznámkovém bloku nebo základním textovém editoru.

1.2.1.1 Elementy

Elementy definují strukturu a vzhled výsledné stránky. Většina z nich je „párová“, tvoří ji tedy počáteční a koncová značka. Díky tomu lze jednotlivé elementy do sebe vnořovat a tím tvořit komplexní struktury, definující vzhled a funkce stránky.

Jednotlivé elementy lze také dále upravovat. Většina umožňuje přidání parametrů, jako je identifikátor elementu, styly nebo třídy. Ty definující vzhled komponenty, pozici v rámci stránky, případně nějakou rozšiřující funkcionalitu daného komponentu. Některé elementy mají definovány i své vlastní parametry, u tlačítek/odkazů se jedná například o cestu k dané stránce, u obrázků jde o odkaz na použitý zdroj.

Co se týče obsahu elementů, většina z nich dokáže v nějaké podobě zobrazovat text, který je v nich obsažen. Některé jsou ovšem přímo určeny pro zobrazení textu, který podle definovaných parametrů upravují. Jako příklad zde lze uvést <h1>, který zobrazuje text jako nadpis s velikostí určenou číslem v elementu, nebo <p> sloužící k zobrazení paragrafu textu. [9]

1.2.1.2 Struktura stránky

Hlavním prvkem, který zahrnuje všechny další elementy v rámci HTML stránky, je element <html>. Ten následně obsahuje další komponenty, jako je <head> a <body>. Hlavičková část obsahuje informace o stránce, tedy titulek, metadata o stránce a propojení s externími zdroji, jako jsou fonty, kaskádové styly nebo skripty použité na stránce. V druhém řečeném elementu je potom obsažen samotný vzhled a prvky stránky, jako jsou texty, tlačítka, tabulky a další. [9]

1.2.2 CSS

Kaskádové styly slouží k definování a upravování vzhledu webových HTML stránek [9]. Lze pomocí nich kontrolovat vzhled jak jednotlivých komponentů, tak i stránky jako celku. Tyto styly mohou být definovány a uloženy přímo v HTML souboru, nebo, pro lepší přehlednost, v externím souboru, ke kterému se přidává reference v hlavním souboru stránky.

CSS je jazyk založený na pravidlech. Uživatel definuje parametry, které by měly být aplikovány na určitý element nebo skupinu na webové stránce. Těchto parametrů je celá řada, od těch, co definují barvu a velikost fontu, až po zarovnání v rámci stránky a nastavení viditelnosti elementu. [10]

Kromě přímého určení, na jaký element by se měl styl aplikovat, je možné vytvořit vlastní třídu s definovanými parametry, který lze posléze aplikovat na libovolný element tím, že se přidá název třídy do „class“ sekce elementu. Za předpokladu, že jsou styly dostupné a že příslušný element dokáže na poskytnuté parametry reagovat, se styl jako takový projeví.

1.2.3 SQL

Pod touto zkratkou se skrývá název Strukturovaný Dotazovací Jazyk [11]. Ten slouží k přistupování a manipulování s databázovými systémy a databázemi samotnými. I přesto, že se jedná o standard, existují různé varianty tohoto jazyka, většinou určené pro konkrétní databázový systém. Většina hlavních příkazů je ovšem mezi variantami stejná. [11]

1.2.3.1 Příkazy

Příkazy tohoto jazyka lze rozdělit na několik skupin, založených na jejich charakteristice. První skupinou jsou „definice dat“, do kterých spadají příkazy jako CREATE pro vytvoření tabulek a dalších objektů, ALTER, určený pro modifikaci databázových objektů a DROP, sloužící pro mazání objektů.

Druhá skupina se používá pro „manipulaci s daty“. K tomu slouží příkazy SELECT pro vybrání záznamů z jedné nebo více tabulek, INSERT pro vkládání dat, UPDATE, sloužící k aktualizaci dat a DELETE pro mazání dat z tabulek.

Poslední skupina je určena pro „kontrolu dat“. To zahrnuje příkazy GRANT, který přiděluje uživateli práva na manipulaci s daty a přístupu k datům samotným, a REVOKE pro odebírání těchto práv. [11]

1.2.4 XML

Extensive Markup Language je značkovací jazyk, podobný HTML, ovšem bez předdefinovaných značek či tagů [12]. Uživatel si zde definuje vlastní značky, čímž si strukturu a obsah dokumentu psaném v tomto jazyce může libovolně přizpůsobovat. Tento formát se používá se jako jedna z možností, jak ukládat, hledat či sdílet data. Výhodou je jeho standardizace, což umožňuje jeho sdílení a přenášení mezi platformami a systémy, bez nutnosti dokument nějak měnit.

Stejně jako u HTML, celková struktura je definována pomocí skládání tagů, které tvoří syntaxi jazyka, do objektů s nějakým významem. Jako příklad lze uvést objekt se zprávou, který obsahuje „podkategorie“, jako je nadpis, tělo zprávy nebo autora, od kterého zpráva přišla. Jednotlivé části lze potom dále dělit, ku příkladu tělo lze rozdělit na pomyslný úvod zprávy, samotné tělo a nakonec pozdrav.

Jazyk samotný nemá definovaný „základní“ vzhled, potažmo se primárně nezabývá vzhledem výsledné reprezentace, pouze samotnými daty. Proto je hlavně používaný na přenos dat, případně i pro ukládání dat. Přesto existují možnosti, jak vzhled definovat a data zobrazit v upraveném stylu, například za použití kaskádových stylů.

Kromě přenosu dat se jazyk XML využívá třeba pro ukládání vektorových obrázků ve formátu SVG, v rámci dokumentů pro kancelářské aplikace typu Office, pro zápis notové stupnice nebo matematických vzorců pomocí modifikovaných verzí tohoto jazyka. [12]

1.2.5 JSON

Pro přenos dat se využívá také tzv. JavaScript Object Notation [13], používaný jako nativní formát pro JavaScript aplikace. Je jednoduchý, co se týče jak čitelnosti pro běžného uživatele, tak i pro generování a syntaktickou analýzu ze strany softwaru. Přesto, že byl původně součástí standardu jazyka JavaScript, je dnes platformě nezávislý a lze ho používat, podobně jako XML, na většině moderních systémů. Používanými konvencemi je velmi podobný jazykům z rodiny C, jako je C, C++, C#, Java, Python atd.

Celý jazyk je založen na dvou typech univerzálních struktur. Prvním je kolekce, skládající se z dvojice název a hodnota. Ve spoustě jazyků je toto realizováno jako objekt, záznam, struktura nebo třeba hash tabulka, případně asociativní pole. Druhou strukturou je pole hodnot. Ve většině jazyků je tohle reprezentováno pomocí pole, vektoru, seznamu či sekvence. V podstatě všechny moderní jazyky tyto objekty v nějaké podobě podporují, proto dává smysl, že tento formát je zaměnitelný v rámci programovacích jazyků, obsahující podobné struktury. [13, 14]

1.2.5.1 Základní syntaxe

Základním tvarem dat jsou objekty. Jedná se o neseřazené párové dvojice název a hodnota, ohraničené složenými závorkami. Každý název je od hodnoty oddělen dvojtečkou a jednotlivé páry od jednotlivých dvojic čárkou. Pole je seřazená kolekce hodnot, oddělených čárkami a jako celek ohraničeny hranatými závorkami.

```
[
  {
    "id": "08da2127-ee9f-40b3-88e4-741ee3ea1aaf",
    "name": "Household_1",
    "description": "First household",
    "dateCreated": "2022-04-18T10:41:01+00:00"
  }
]
```

Obrázek 2 JSON příklad objektu

Jazyk rozlišuje řadu typů hodnot, které existují i ve většině programovacích jazyků. Jedná se o textové řetězce, čísla, pole, booleovské hodnoty, prázdné hodnoty a pole. Textové řetězce mohou obsahovat kromě samotného textu i další speciální znaky, například označení konce řádku, uvozovky a další. Čísla také mohou být v různých tvarech, mohou to být

například desetinná čísla, záporná čísla, ale také i velká čísla za použití vědecké notace (např. 1.5e+10). [13]

1.3 Frameworky a nástroje

Obecně se pod pojmem framework skrývá reálná nebo konceptuální struktura, určená jako podpora nebo návod pro vytvoření nebo pro rozšíření existující struktury na něco užitečného. V počítačových systémech framework často představuje vrstvenou strukturu, která obsahuje funkce, které řeší některé implementace a umožňují uživateli zaměřit se na vysokoúrovňovou logiku aplikace, místo řešení základních problémů a nízkoúrovňových funkcí. [15, 16]

1.3.1 .NET

.NET je bezplatná, cross-platformní open-source vývojářská platforma, sloužící pro vytváření různorodých aplikací a programů [17]. Lze ho využívat s vícero programovacími jazyky, v mnoha vývojových prostředích a obsahuje nástroje a knihovny pro vývoj webových, mobilních a desktopových aplikací, četně počítačových her a IoT zařízení. [17]

Primárním programovací jazykem, na který je .NET zaměřen, je C#. Výhodou je, že se nejedná o jediný podporovaný jazyk, ale podporuje možnost použití celé řady jazyků, jako je VisualBasic, C++, F#, varianty jazyků Ruby a Python, a další. [18]

1.3.1.1 Historie a názvosloví

Během své existence měl .NET mnoho variant a některá pojmenování s postupem času měnila svůj význam. Prvotní implementací byl tzv. .NET Framework [19], který byl dostupný pouze pro platformu Windows a jeho zdrojový kód zprvu privátní, později částečně přístupný veřejnosti. V této verzi byly definovány základy frameworku jako takového, jeho struktura a jeho komponenty.

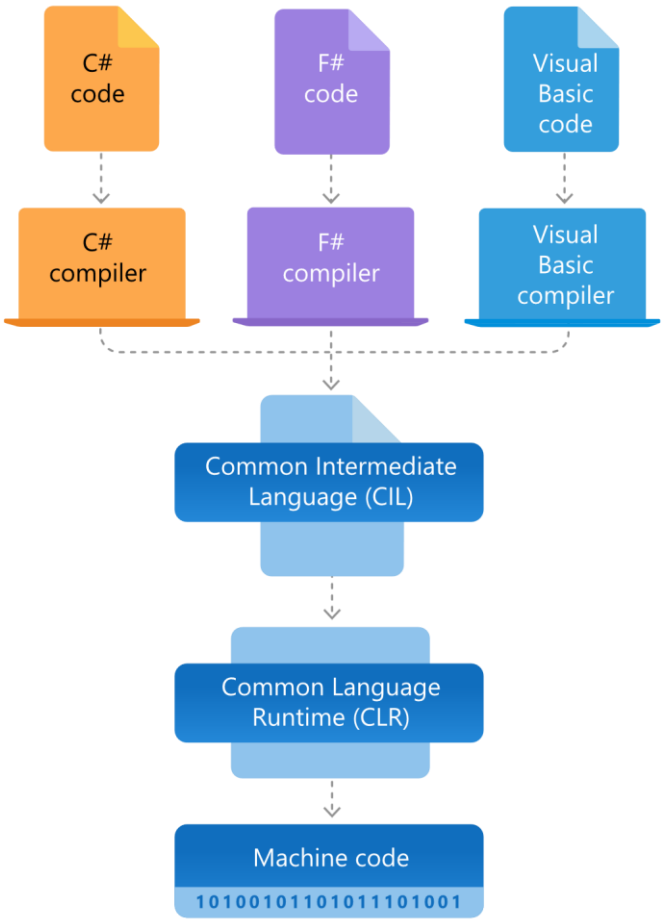
Kvůli vzestupu nových typů zařízení a cloudových služeb došlo ke změně trendů, definující požadavky na vývoj softwaru. Aby se Microsoft přizpůsobil těmto novým technologiím, vytvořil novou variantu frameworku zvanou .NET Core. To byla v podstatě multiplatformní a open-source implementace .NET Frameworku, se kterým sice sdílela základní části, ovšem v mnoha se také lišila. Hlavní důraz byl kladen na samotnou multiplatformitu, která umožňovala vytvářet aplikace, jejichž kód by běžel ve stejné podobě na platformách, jako je x86 nebo ARM, případně také pomocí softwaru Docker [20].

.NET Core se stal pomyslným hlavním projektem, neboť drtivá většina nových inovací a vylepšení se provádí v rámci této verze, která byla v jeho 5. verzi přejmenována na pouze .NET. Verze Framework je sice stále vyvíjena, ale jen velmi pomalu.

Co se týče názvosloví, existuje ještě tzv. .NET Standard [21]. Nejedná se o implementaci, ale o definici, určující, co musí jiné implementace obsahovat za funkce a API. Používá se hlavně pro vývoj knihoven v rámci .NET a existuje ho několik verzí. Je cílen hlavně na starší verze, od .NET 5 výše už většinou není potřebný. [21, 22]

1.3.1.2 Hlavní komponenty

Hlavní součástí .NET je komponenta CLR, neboli Common Language Runtime. Jedná se o tzv. „Virtual Machine“, která vytváří virtuální prostředí, ve kterém běží kód aplikace. Toto prostředí se stará o základní potřeby programů, jako je správa vláken, bezpečnost, správa paměti, robustnost a tak dále. Hlavním účelem je spouštění .NET programů nezávisle na programovacím jazyku a platformě.



Obrázek 3 .NET architektura [19]

Součástí je taktéž Framework Class Library, tedy kolekce znovupoužitelných, objektové orientovaných knihoven a funkcí, poskytující základ pro všechny programy vytvořené v rámci .NET platformy. Lze je přirovnat k hlavičkovým knihovnám v jazycích C a C++. [18, 19]

1.3.1.3 NuGet

Tento nástroj slouží ke sdílení kódu mezi uživateli v rámci .NET platformy. NuGet balíček obsahuje kompilovaný kód, který lze připojit ke vlastnímu kódu za účelem jeho rozšíření o další funkcionalitu.

Samotný nástroj se stará i o možné reference a závislosti. Když si uživatel chce stáhnout nějaký balíček, nástroj uživatele upozorní, že je pro správné fungování potřeba dalších součástí a zeptá se, jestli je má taktéž stáhnout a přidat do projektu.

Výhodou těchto balíčků je to, že mohou být napsány v libovolném podporovaném programovacím jazyku a výsledek bude možné využít v jakémkoliv řešení a projektu, vytvořeném v rámci .NET platformy. [23]

1.3.1.4 LINQ

Language-Integrated Query je nástroj, jehož účelem je integrace dotazovacích příkazů přímo do jazyka C#. Používá se k filtrování, seřazování a seskupování dat z podporovaných kolekcí, jako je pole nebo třeba List.

```
// Specify the data source.
int[] scores = { 97, 92, 81, 60 };

// Define the query expression.
IEnumerable<int> scoreQuery =
    from score in scores
    where score > 80
    select score;

// Execute the query.
foreach (int i in scoreQuery)
{
    Console.Write(i + " ");
}

// Output: 97 92 81
```

Obrázek 4 LINQ příklad syntaxe a použití [24]

Principiálně ho lze přirovnat k jazyku SQL. Oba jsou zaměřeny na výběr dat z kolekce / databáze a využívají podobnou syntaxi a klíčová slova. Oproti SQL je ovšem LINQ nezávislý na typu kolekce a na typu dat samotných. To umožňuje použít ten samý dotaz na různé datové typy a přistupovat k nim stejným způsobem.

Podporu tohoto nástroje má v jazyce C# většina datových typů. Konkrétně se jedná třeba o obyčejné pole, datový typ List, soubor ve formátu XML, případně lze přistupovat „přímo" databázi. Taktéž je zde možnost vytvořit vlastní datový typ, kterému lze poté přidat „podporu" dotazování pomocí LINQ.

Další výhodou je to, že se jedná o „známý" jazyk, kde není potřeba se učit nový způsob dotazování nebo přímo nový jazyk. Taktéž použitím stejné syntaxe, jako má jazyk samotný, se zjednodušuje a zpřehledňuje kód. LINQ samotný také přímo zahrnuje kontrolu objektů, čímž do určité míry předchází možným zneužitím a napadením koncové kolekce či databáze. [24]

1.3.1.5 ASP.NET

Tato součást platformy .NET je zaměřena primárně na tvorbu webových aplikací [25]. K tomuto účelu obsahuje potřebné nástroje, jako je syntaxe pro vytváření webových stránek pomocí jazyka C#, knihovny zahrnující běžně používané webové návrhové vzory, třeba MVC, a další užitečné funkce, například systém pro ověřování uživatelů Identity [26].

Atraktivnější verzí tohoto rozšíření je ASP.NET Core, který podobně jako .NET samotný open-source a multiplatformní. Tato verze je aktuálnější a je na ni primárně zaměřen vývoj samotný. [25]

1.3.1.6 Razor pages

Razor stránky slouží pro tvorbu webových stránek, u kterých je dynamická a interaktivní část tvořena pomocí kódu psaném v jazyce C# místo běžně používaného JavaScriptu. Tyto soubory kombinují HTML kód, určující vzhled stránky, a logiku, která reaguje na požadavky. Tyto části mohou být buďto v jednom společném souboru nebo odděleně ve dvou různých, kde soubor obsahující logiku je více podobný standartním třídám v rámci jazyka C#. Tento typ stránek využívá i Blazor, který je popsaný v následujíc části.

1.3.2 Blazor

Technologie Blazor [27] je v době psaní této práce stále poměrně nová. S využitím Razor stránek umožňuje vytvářet interaktivní webové aplikace, které místo jazyku JavaScript využívají pro psaní logiky jazyk C#, čímž odpadá nutnost učit se více jazyků pro tvorbu komplexních webových aplikací.

Blazor aplikace se skládají ze znovupoužitelných komponentů psaných pomocí kombinace HTML, CSS a C#. Jak klientská, tak serverová část může tento kód sdílet a používat. Komponenty obsahují flexibilní UI logiku a zpracovávají uživatelské události, a také mohou být vzájemně skládány a znovu používány. Také je zde možnost předávání údajů a dat mezi jednotlivými komponenty, jak pomocí vlastností, tak i například pomocí funkcí.

Kromě funkcionality v jazyce C# je možné implementovat i JavaScriptové funkce a knihovny, pomocí tzv. JavaScript Interop. Tyto funkce je pak možné volat v C# kódu, a naopak funkce napsané v C# je možné používat v JavaScriptové implementaci.

1.3.2.1 Varianty

V současnosti existují dvě verze této technologie. I přes jejich podobnost se v některých klíčových vlastnostech a implementaci liší. Kromě dále zmíněných rozdílů se jedná například o způsob vykreslování, komunikace jednotlivých částí a některých dalších implementací.

První varianta pracuje na straně klienta, k čemuž využívá technologii WebAssembly. Celá aplikace, četně všech závislostí, je odesílána ke klientovy, který ji spouští lokálně přímo u sebe. Výhodou této varianty je možnost nasadit aplikaci bez serverové části, tedy jako „standalone“ aplikaci, která vykonává veškeré operace přímo u klienta. V případě kombinace se serverovou částí je aplikace samotná taktéž do jisté míry nezávislá, potažmo nezáleží na tom, v jakém jazyku nebo technologii je serverová část vytvořena. Výhodou je taktéž to, že klient nemusí mít na svém zařízení naistalovány webový server nebo ASP.NET Core knihovny, proto je možné aplikaci spustit ve všech moderních prohlížečích. Nevýhodou této varianty je hlavně omezená funkcionalita prohlížeče a velikost stahovaných dat, protože dochází k přenosu celé aplikace a jejích částí.

Druhá verze fyzicky je propojena se serverem a je na něm závislá. Komunikace je realizována pomocí frameworku SignalR, který umožňuje posílat zprávy v reálném čase mezi oběma stranami komunikace. Výhodou tohoto přístupu je potřeba přenosu menšího počtu dat mezi klientem a serverem, čímž jsou sníženy požadavky na síť a koncové zařízení.

Taktéž je v tomto případě aplikace bezpečnější a umožňuje využít více funkcí, ke kterým by za použití první varianty neměla přístup. Záporem u tohoto přístupu jsou větší požadavky na hostující server v případě vyššího zatížení, vyšší latence nebo také úplné ztracení funkcionality v případě přerušení spojení.

V současnosti je vyvíjena i hybridní verze, kombinující výhody obou přístupů. Tato varianta je ovšem dostupná v preview verzi, která se může oproti finální verzi velmi lišit, proto není vhodné ji v současnost využívat pro vývoj aplikací, určených pro reálné použití. [27]

1.3.3 Wiring

Wiring [28] je open source framework určený pro programování mikrokontrolerů. Umožňuje psát cross-platformní software pro celou řadu mikrokontrolerů a zařízení, které k nim lze připojit. [28] Jeden z takových mikrokontrolerů je i Arduino, populární jednodeskový počítač nebo řada Wi-Fi modulů ESP8266.

Jednotlivé programy se nazývají „Sketches“, což má symbolizovat proces rychlého prozkoumávání a iterace nápadů, ze kterých se ty nejzajímavější dále upravují a následně implementují v iterativním procesu vývoje. [29]

1.3.3.1 Části

Samotný Wiring se skládá ze tří hlavních částí. První je Wiring Language, což je vrstva nad programovacím jazykem C++, která definuje některé nové funkce a zjednodušuje proces tvorby programů pro embedded zařízení a mikrokontrolery.

Druhou částí je Wiring Framework. Ten používá C/C++ API pro psaní programu, buď pomocí jazyka Wiring nebo přímo jazyka C++. Taktéž je možné využívat knihovny a kód psaný přímo v jazycích C a C++.

Poslední částí je Wiring IDE, tedy vývojové prostředí sloužící pro psaní, kompilování a nahrávání programů do podporovaných zařízení. [28] Samotné prostředí je napsáno v jazyce Java a vychází z prostředí vytvořeného pro programovací jazyk Processing. V dnešní době je asi nejznámější variantou Arduino IDE, soužící k programování nejen stejnojmenných zařízení, ale i dalších zařízení. [29]

1.3.4 Entity framework

Tento open-source nástroj [30] slouží pro mapování databázových objektů na objekty v rámci .NET aplikací. Mapováním je myšleno propojení proměnné v jednom objektu na

proměnnou v objektu druhém. Umožňuje vývojářům pracovat s daty pomocí .NET objektů bez nutnosti mít povědomí o vnitřní struktuře databáze a způsobu uložení dat. Taktéž odstraňuje potřebu znalosti nebo tvorby kódu, který k datům přistupuje a jak s nimi dále pracuje.

Původně byl framework součástí standartu .NET, v pozdějších verzích byl od celku oddělen a momentálně existuje jako samostatný nástroj. V současnosti existují dvě verze tohoto frameworku, a to EF6 a EF Core. Rozdílově jsou podobné verzím .NET samotného. EF 6 je určen pouze pro operační systém Windows, kdežto verze Core je multiplatformní a je na něj primárně zaměřen vývoj.

Entity framework dokáže pracovat jak s přístupem Code-First, kdy nejdřív je v rámci aplikace vytvořena struktura objektů, která je poté aplikována na databázi samotnou, tak jako Database-First, což se v podstatě opak přechozího řečeného.

V rámci struktury aplikace se Entity Framework nalézá mezi databází a business objekty, alternativně pojmenované jako business modely nebo doménové třídy. Data z databáze jsou ukládány do proměnných v rámci business objektů, se kterými se v aplikaci dále pracuje. Tento převod se provádí automaticky, přesto jde některé vztahy definovat ručně, případně změnit nebo úplně zrušit.

Co se funkcionality týče, framework umožňuje generovat databázi jako takovou, přistupovat k datům pomocí transakcí, má ochranné mechanismy pro zachování celistvosti dat a také zabudované základní konvence pro jednotlivé používané návrhové a implementační vzory. Vytváření a aktualizování databází je v případě Code-First přístupu prováděno pomocí tzv. migrací, obsahující seznam změn oproti předchozí verzi databáze. [30]

1.3.5 Frameworky a sady kaskádových stylů

Vzhled webových stránek je tvořen pomocí kaskádových stylů, které určují tvar, barvu a další parametry jednotlivých částí HTML kódu. Pro ulehčení tvorby webů lze využít různé sady stylů, které mají již definovány základní vlastnosti a vývojář je pouze skládá dohromady podle svých potřeb.

1.3.5.1 Bootstrap

Bootstrap [31] je zdarma dostupný frontendový framework pro vytváření webových stránek a aplikací. Kromě vzhledů poskytuje i funkcionalitu pro některé komponenty a možnost vytvářet responzivní stránky, které se přizpůsobují velikosti a orientaci zařízení. Pro

pokročilejší vlastnosti a efekty některých elementů je zde využíván jazyk JavaScript, případně nějaká jeho nastavba.

Obsahuje rozsáhlou řadu základních elementů, které si může uživatel přizpůsobit a libovolně kombinovat. Jednotlivé objekty lze pomocí něho například centrovat v rámci stránky, obarvit, změnit jejich „vrstvu“, přidat jim odsazení, změnit velikost a další. Většina z nich má také různé animace přechodů a výběrů, některé dokonce i specifickou funkcionalitu a slouží například jako navigační komponenty, notifikace a třeba formuláře. [31]

1.3.5.2 TailwindCSS

Podobně jako Bootstrap, i TailwindCSS [32] je framework sloužící pro tvorbu vzhledů webů. Oproti Bootstrapu ovšem neobsahuje přímo definované komponenty, jako jsou tlačítka a podobné objekty. Místo toho poskytuje spoustu parametrů a tříd, pomocí kterých si uživatel všechny elementy stránek definuje sám.

Framework samotný také zahrnuje pár dalších užitečných nástrojů. Umožňuje například během sestavování aplikace promazávat nepoužívané třídy a parametry, čímž zmenšuje velikost finálního CSS souboru, definujícího vzhled stránky. Taktéž je zde možnost rozšiřovat již existující parametry o vlastní varianty, třeba přidáním nové barvy, definicí parametru určujícího velikost prvku, či třeba zahrnutím jiných textových fontů.

I přes to, že TailwindCSS samotný nezahrnuje předdefinované komponenty, existuje celá řada volně dostupných vzorů a příkladů, které lze při tvorbě vlastních verzí využít či přímo použít. Výhodou je také existence tzv. TailwindPlay [33], což je oficiální online nástroj, ve kterém si lze vytvářet vzhled stránky v reálném čase a výsledek použít přímo ve vyvíjené aplikaci. [32]

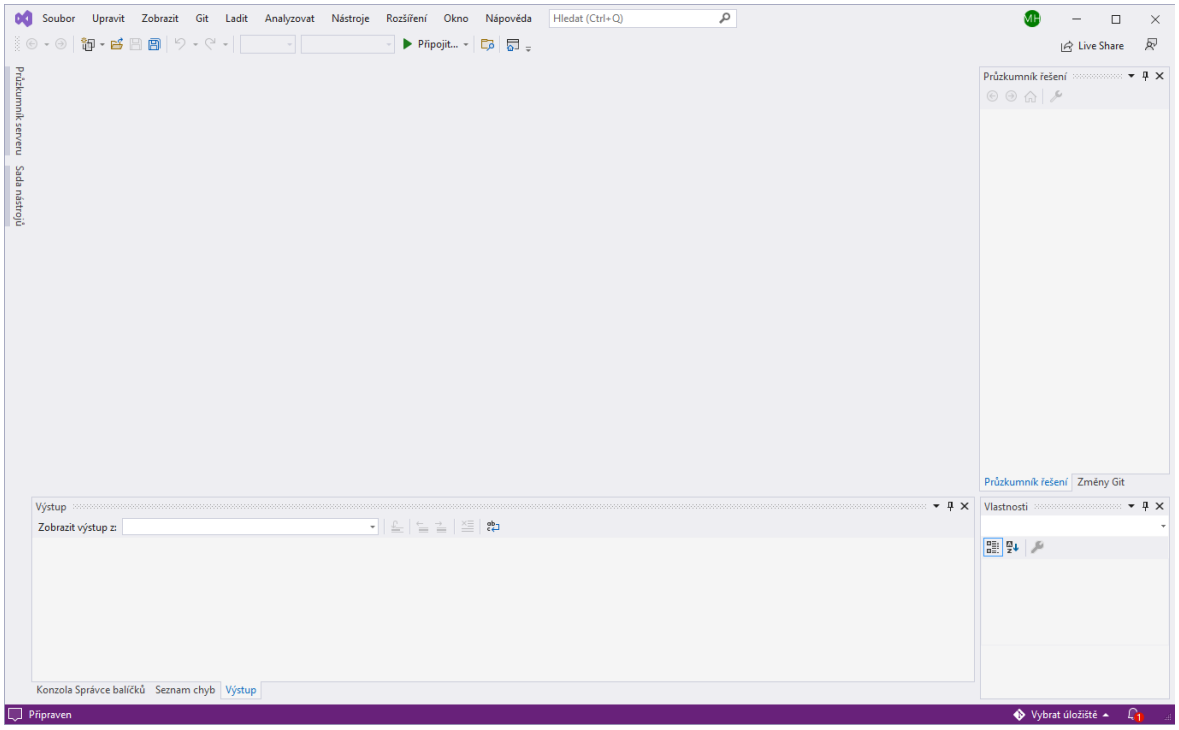
1.4 Integrované vývojové prostředí

Pro vývoj aplikací a programů je vždycky výhodou mít k dispozici nástroje, které tento proces usnadňují. Jako nejdůležitější můžeme pokládat samotné vývojové prostředí, tedy program určený pro tvorbu dalších programů. Programovat lze samozřejmě i v jednoduchém textovém editoru, ovšem výhody plnohodnotného prostředí, jako je zvýrazňování syntaxe či našeptávání příkazů, nelze zapřít.

1.4.1 Visual Studio

Visual Studio [34] je vývojové prostředí vyvinuté společností Microsoft pro tvorbu různorodých aplikací za použití celé řady podporovaných jazyků. Podporuje také rozličné frameworky, jako je primárně .NET, dodatečné nástroje pro vývoj, například verzovací systém Git a správce balíčků NuGet, a i možnosti stahování řady rozšíření, poskytující další funkce. Prostředí samotné je také možné déle přizpůsobovat, například volbou barvy prostředí, textu a zvýrazněné syntaxe programovacího jazyka.

Prostředí má v současnosti tři verze, lišící se v počtu dalších funkcí a také ceně jednotlivých licencí. Community edice je základní varianta, která je bezplatná a obsahuje základní potřebné nástroje pro vývoj. Licenci na tuto verzi je možné využít do určitého počtu zařízení a zisku, poté je nutné koupit si vyšší verzi. Dalšími edicemi je Professional a Enterprise, každá s dalšími rozšířeními a dodatečnými funkcemi oproti bezplatné verzi. Opět jsou zde různé podmínky, které určují nárok na danou licenci. [34]



Obrázek 5 Visual Studio IDE

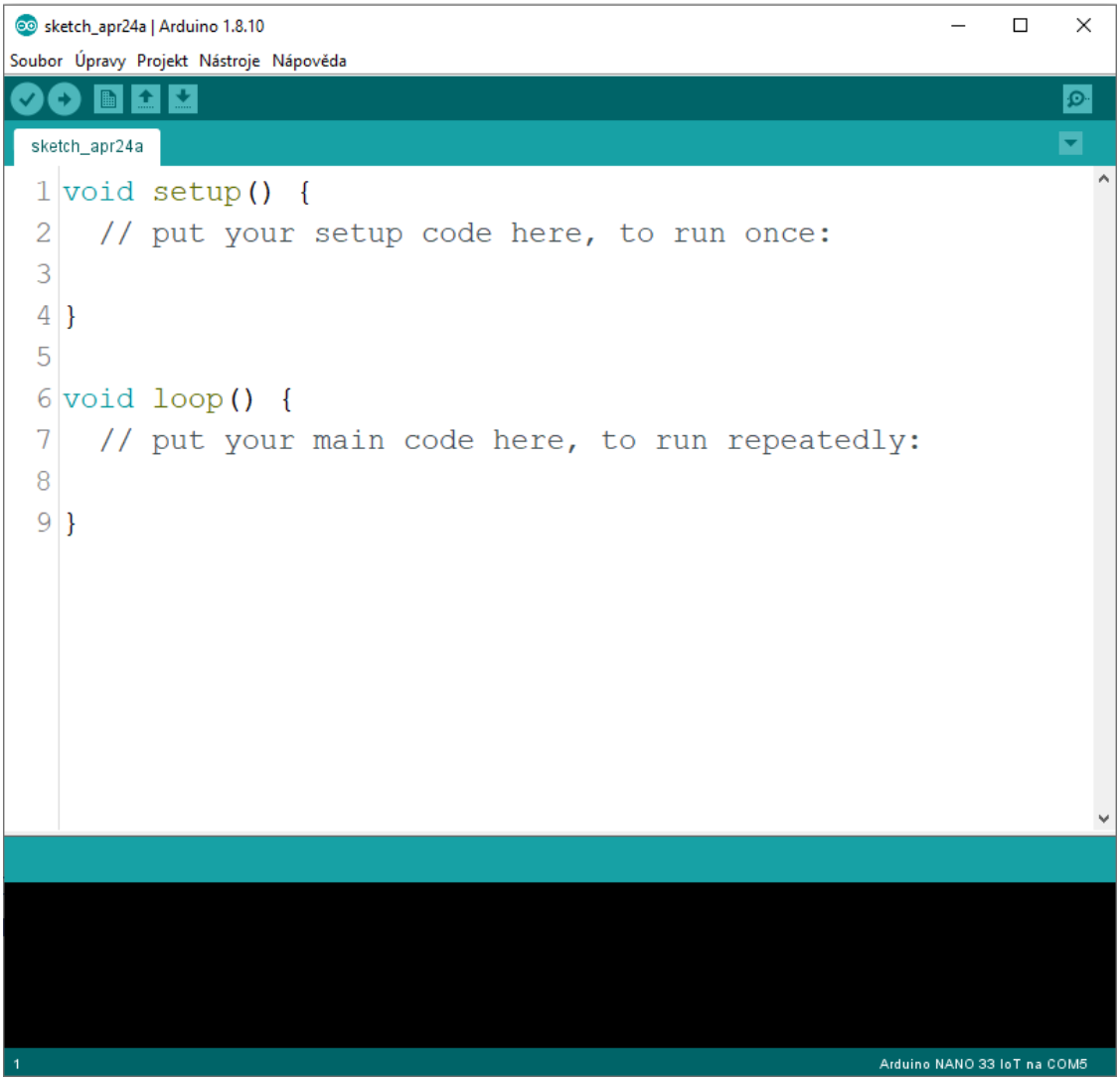
V současnosti existuje i bezplatná, open-source varianta tohoto prostředí, pojmenovaná Visual Studio Code. Výhodami tohoto prostředí je jeho multiplatformní implementace, které je nenáročné na systémové prostředky a umožňuje vývoj ve více jazycích a technologiích. [35] Také díky jeho otevřenosti ho lze využít jako základ pro vlastní prostředí, jako příklad lze uvést novou verzi prostředí Arduino IDE, popsané dále

1.4.2 Arduino IDE

Pro programování mikrokontrolerů, jako je Arduino aj., je používáno prostředí Arduino IDE [36]. To je určeno pro vývoj v programovacích jazycích C/C++ a platformě Wiring. Programy vytvořené v tomto prostředí jsou pojmenovány jako „Sketches“. Prostředí zahrnuje celou sadu nástrojů a již existujících řešení, které stačí pouze nahrát do příslušného zařízení a spustit.

Zahrnuta je také podpora celé řady různorodých mikrokontrolerů a zařízení. Dodatečné definice zařízení a také další knihovny s funkcemi je možné stáhnout z internetu pomocí jednoho ze zahrnutých nástrojů.

Prostředí umožňuje také přímé nahrávání programu do zařízení pomocí sériového připojení, většinou přímo přes USB kabel. Taktéž podporuje zobrazení sériové komunikace se zařízením, pomocí textového sériového monitoru. [36]



Obrázek 6 Arduino IDE

V současné době je ve finálních fázích vývoje nová verze prostředí, které využívá jako základ Visual Studio Code. Tato verze obsahuje nástroje z předchozí verze, obalené v moderním a lépe vypadajícím prostředí.

1.5 Databáze

Databáze je organizovaná kolekce strukturovaných informací nebo dat, které jsou uloženy v elektronické podobě na počítačovém systému. Operace s daty je většinou kontrolována pomocí nějakého systému pro správu databázového systému (DBMS). Data, dohromady s tímto systémem a s nimi spojenými aplikacemi, jsou označovány jako databázový systém, zkráceně nazývané pouze jako databáze.

Možností, jakým způsobem data ukládat, je celá řada, každý zaměřen na jiný typ dat a využití. U relačních databázích jsou data ukládána do navzájem propojených tabulek, tvořených sloupci a řádky. Tento přístup umožňuje jednoduchý přístup k datům, jejich správu, modifikaci a organizaci. Většina databázi využívá pro přístup k datům a dalším operacím jazyk SQL.

Jako pomyslný protiklad relačního přístupu lze považovat databáze typu NoSQL, znamenající „ne jenom SQL“. Tento typ je používán hlavně pro ukládání velkého množství dat, tzv. Big Data, u kterých je potřeba možnost škálovat samotný systém, co se týká jak počtu, tak kvality zařízení. Dalšími příklady mohou být grafové databáze, dokumentové databáze nebo kombinace různých přístupů, nazývané jako multimodelové databáze. [37]

1.5.1 Zástupci

Ke každému typu databází existuje spousta konkrétních implementací, většinou vzájemně velmi podobné. U relačního typu databází jsou jedny z nejvíce běžně používanějších systémů, například MySQL, Microsoft SQL Server nebo PostgreSQL. U NoSQL to jsou systémy, jako je Apache Cassandra, MongoDB či CouchDB. Cloudové databáze mají zastoupení ve formě Microsoft Azure SQL Database a Amazon Relational Database Services. [38]

1.5.2 Výzvy

Na databázový systém jsou kladeny důležité požadavky, zaručující dostupnost a kvalitu poskytovaných služeb. Mezi tyto výzvy patří možnost zpracovávat velké množství dat a požadavků v rozumném čase, požadavky na zabezpečení dat, schopnost systému růst nebo zajištění konzistence a správnosti dat. [37]

2 HARDWARE

Součástí jakéhokoliv softwaru je vždy i nějaký hardware, na kterém je spuštěn. Může se jednat u zařízení z různými možnostmi výkonu a schopností, od víceúčelových zařízení, schopných mít spuštěno několik desítek programů a aplikací zároveň, po jednoduché zařízení z jedním aktivním programem a konkrétním zaměřením.

2.1 Mikrokontrolery a jednodeskové počítače

Mikrokontroler je kompaktní integrovaný obvod, navržený tak, aby dokázal vykonávat specifické operace v tzv. embedded systému. Typicky zahrnuje mikrokontroler procesor, paměť a vstupy / výstupy, integrované v rámci jednoho čipu. Mikrokontrolery lze nalézt celé řadě zařízení, od vozidel a mobilní telefony, až po automaty nebo domácí spotřebiče.

V podstatě se jedná o jednoduchý miniaturní osobní počítač navržený pro ovládání nějaké části v rámci většího zařízení nebo větší komponenty bez komplexního operačního systému. [39]

V této práci bude na jednodeskové počítače a embedded systémy odkazováno obecně jako na mikrokontrolery, i když se technicky jedná pouze o součást většího celku.

2.1.1 Součásti

Mikrokontroler samotný je složen z řady klíčových komponent, které umožňují jeho fungování. Jako nejdůležitější část se dá považovat procesor, tedy pomyslný mozek celého zařízení. Jeho účelem je přijímat a zpracovávat různé instrukce, aritmetické a logické funkce, četně operací týkající se vstupů a výstupů. Také se stará o komunikaci s ostatními částmi embedded systému.

Další klíčovou části je paměť, která se používá pro ukládání dat, které procesor přijímá a používá. Obvykle se používají dva typy pamětí. Prvním typem je paměť dlouhodobá, která má v sobě uloženy informace o instrukcích, které může procesor vykonávat, četně programu, podle kterého vykonává mikrokontroler svou činnost. Druhou je paměť určená pro data, která jsou používána během vykonávání programu a instrukcí. Oproti prvnímu typu je tento typ paměti většinou rychlejší, ovšem na úkor nutnosti napájení pro udržení dat, které obsahuje. Takovým pamětem se také říká volatilní, tedy závislé na proudu, oproti prvnímu typu, který je pojmenován nevolatilní.

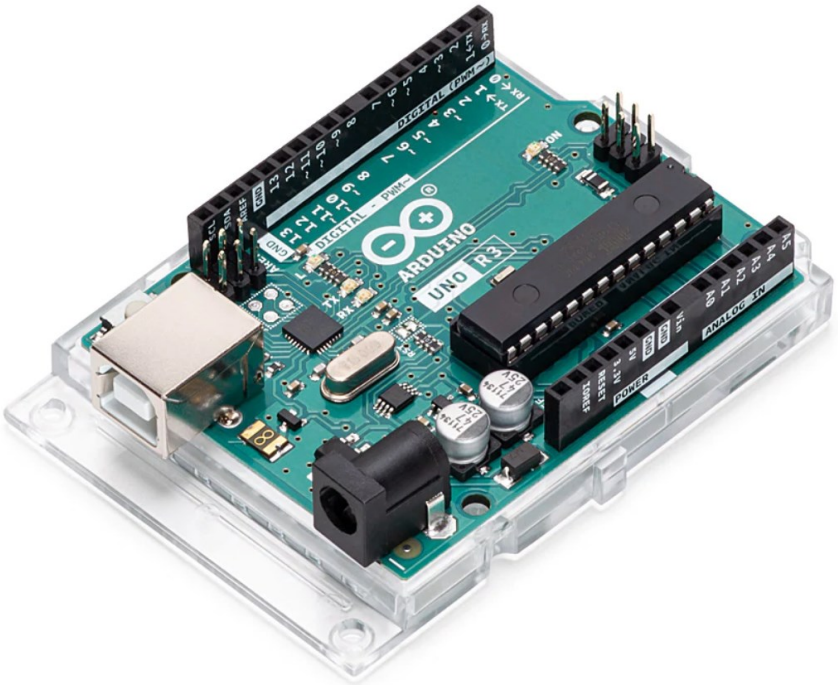
Poslední důležitou částí jsou vstupní a výstupní periferie. Ty slouží jako rozhraní mezi procesorem a okolním světem. Vstupy posílají procesoru informace, on na ně reaguje a odpovídá pomocí výstupů.

Případné další součásti už nejsou pro fungování mikrokontroleru kritické, ale přesto užitečné. Zde můžeme zahrnout převodníky, starající se o převod analogových vstupů na digitální a naopak, systémové sběrnice pro propojení více zařízení pomocí jednoho spojení, nebo sériové porty, sloužící pro připojení externích zařízení, například pomocí USB portu. [39]

2.1.2 Konkrétní příklady

2.1.2.1 Arduino

Arduino [40] je open-source elektronická platforma založená na hardwaru a softwaru, který je jednoduchý na použití. Tyto desky umožňují čtení analogových i digitálních vstupů z různých senzorů či zařízení, data zpracovat a výsledek poslat na výstup. Tvorba programu je prováděna za použití již zmíněného frameworku Wiring, programování je tedy prováděno pomocí nadstavby jazyka C++, případně tohoto jazyka samotného. Platforma je zaměřena na výuku programování, umožňuje jednoduchá prototypování složitějších zařízení a také slouží lidem pro vytváření vlastních zařízení a projektů.



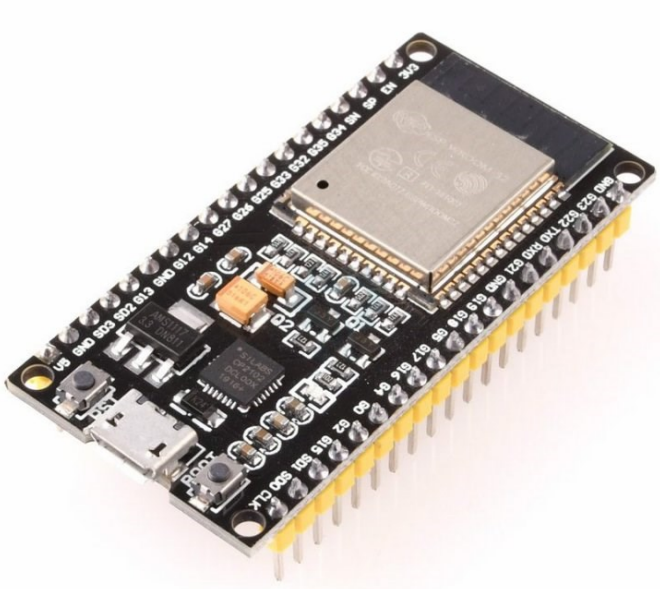
Obrázek 7 Arduino UNO Rev3 [41]

Existuje několik variant a druhů těchto desek, lišící se velikostí, funkcemi či výkonem. Od nejmenšího Arduino Nano a jeho variant, přes desky UNO až po největší a nejvýkonnější DUE. Funkcionalitu většiny těchto mikrokontrolerů lze rozšířit také pomocí tzv. Shieldů, tedy desek s obvody, které se přímo připojují na výstupy hlavních deset a přidávají například ovladatelná relé, elektroniku na ovládání různých typů motorů nebo možnost připojení k internetu pomocí ethernet kabelu nebo Wi-Fi. [39]

Kromě oficiálních variant existují díky otevřenosti platformy i „kopie“ desek, buďto v některých oblastech sofistikovanější, nebo ve většině případů levnější a „ořezané“ varianty originálu.

2.1.2.2 ESP8266 a ESP32

Řada modulů ESP jsou levné Wi-Fi moduly, vhodné hlavně pro různé projekty DIY v rámci Internetu věcí a automatizace domácnost. Oproti jiným mikrokontrolerům, jako je Arduino, jsou především varianty ESP32 o něco výkonnější, ovšem jejich hlavní předností je to, že všechny mají integrované komponenty pro bezdrátovou komunikaci. Pro programování se běžně využívá stejný software a jazyk, jako u Arduina, tedy Wiring a Arduino IDE, navíc ovšem tento mikrokontroler podporuje programování i v jazyku MicroPython, což je implementace jazyku Python pro mikrokontrolery a embedded zařízení. [42]



Obrázek 8 ESP32 [43]

2.2 Server

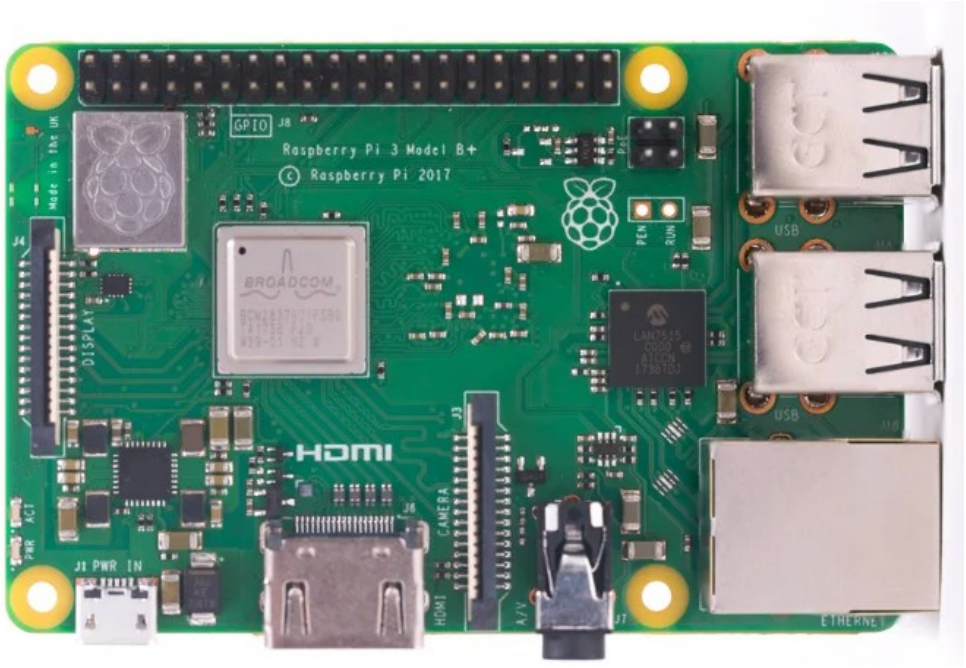
Softwarový systém, jehož požadavkem je neustálá dostupnost, potřebuje pro splnění tohoto požadavku dedikované zařízení, na kterém bude systém běžet, nazývané server. Ten může poskytuje kromě nepřetržité dostupnosti dané služby i možnost přistupovat a využívat ji i z jiných zařízení, než jsou ta přímo připojená k tomuto serveru nebo ke stejné lokální síti.

Jako server lze použít většinu počítačových systémů, od jednoduchých jednodeskových počítačů, až po komplexní serverové zařízení, velikostně zahrnující klidně i místnosti. Takové zařízení nemusí být přímo vlastněno uživatelem, ale může si ho pronajímat, případně využívat jako službu.

2.2.1 Raspberry Pi

Raspberry Pi [44] je pojmenování řady malých jednodeskových počítačů, jejichž cílem je vzdělávání lidí v počítačích a počítačových technologiích. Od vypuštění originální verze v roce 2012 vzniklo již několik iterací a variant, vedoucí až k nejaktuálnější variantě Raspberry Pi 4.

Jako operační systém využívá modifikovanou Linuxovou distribuci Debian, pojmenovanou Raspbian, která má v sobě zahrnutou podporu hardwaru přítomného na desce, jako jsou piny typu GPIO, označující univerzální vstupy a výstupy. Nejedná se ovšem o jedinou volbu operačního systému a uživatel může do zařízení nahrát libovolnou distribuci, případně i variantu Windows určenou pro ARM procesory.



Obrázek 9 Raspberry Pi 3 Model B+ [45]

Díky otevřenosti a open-source řešení existuje, podobně jako u Arduina, řada variací a klonů této desky, lišící se zaměřením, cenou a funkcionalitou.

Jak již bylo řečeno, hlavním zaměřením zařízení je poskytnutí možnosti vzdělávání v oblasti počítačů, například programovaní nebo naučení se zacházek se zařízením s Linuxovou distribucí. Počítač je také vhodný na tvorbu projektů zaměřených na automatizaci domácnosti, jako je třeba zapínání a vypínání světel jako reakce na nějaký podnět, třeba příchod uživatele domů z práce. Je také možné použít tento počítač jako domácí server, který hostuje různé programy a služby, jejichž účelem je třeba právě automatizace domácnosti. [45]

2.3 Další hardware / sensory

Systém popisovaný v této diplomové práci využívá i další hardware, přímo i nepřímo. Jako přímo používané zařízení jsou myšleny hlavně senzory, používané v rámci meteostanice. Ty slouží k zaznamenávaní různých vlastností okolí a předávání těchto informací ke zpracování a následnému odeslání. Konkrétní senzory a zařízení v této kategorii jsou podrobněji popsány v praktické části práce.

Nepřímo použitým hardwarem jsou zařízení, které přispívají ke správné funkcionalitě systému, ovšem nejsou jeho součástí. Do této kategorie lze zahrnout hardware a software starající se o internetovou infrastrukturu, tedy Wi-Fi přístupové body, routery, modemy a další síťová zařízení. Nutné je také napájení jednotlivých zařízení, zahrnout se proto zde dají i napájecí zdroje, převádějící napětí z elektrické sítě na použitelné napájení pro zařízení.

3 DALŠÍ TECHNOLOGIE

Kromě již popsaných příkladů hardwaru a softwaru ještě existují jiné technologie, které se do předchozích sekcí tematicky nehodily. V této části budou popsány koncepty a pojmy, které jsou nějakým způsobem spojeny s vývojem webových aplikací a konkrétního systému tvořeného v rámci této diplomové práce. To zahrnuje hlavně technologie na přenos, zabezpečení a zobrazení dat a informací na webových aplikacích a stránkách.

3.1 IoT

Pojem Internet věcí popisuje síť fyzických zařízení, konkrétně třeba různých senzorů nebo zařízení, vzájemně propojených za účelem předávání a výměny dat přes tuto síť. Může se jednat o různorodá zařízení, o obyčejných předmětů v domácnosti až po sofistikované průmyslové nástroje.

Hlavní výhodou této technologie jsou nízké náklady, jelikož jednotlivé senzory jsou velmi levné a spolehlivé. Navíc jsou vyráběné celou řadou výrobců, díky čemuž existuje velký výběr možných alternativ z rozdílnými parametry a zaměřením. Internet věci může být také použit v kombinaci se strojovým učením, umožňující jednodušší získávání a zpracovávání získaných dat. [46]

3.2 HTTP

Protokol HTTP [47] je používaný pro získávání zdrojů, jako jsou HTML dokumenty, a je základem většiny výměny dat na internetu. Jedná se o protokol typu klient-server, což znamená, že požadavky jsou vytvářeny na straně uživatele, většinou webového prohlížeče, a následně zpracovávány na straně poskytovatele, myšleno serveru.

HTTP je navržen tak, aby byl co nejjednodušší a zároveň čitelný člověkem, i přes přidanou komplexitu v jeho dalších verzích. Taktéž je jednoduše rozšiřitelný, nové funkce mohou být zavedeny pouze domluvou mezi klientem a serverem. Jedná se o bezstavový protokol, neexistuje proto přímé spojení mezi oběma stranami komunikace, přičemž stavovou komunikaci lze případně napodobit pomocí tzv. HTTP Cookies.

Zjednodušeně řečeno, HTTP požadavek se skládá z několika částí. Konkrétně z použité metody, cesty k požadovanému zdroji, verze HTTP protokolu, volitelně z hlavičky s dalšími informacemi a případně i z těla požadavku, používaného hlavně u metod, jako je POST. Odpovědí na tyto požadavky je opět verze protokolu, stavový kód označující úspěšnost akce,

zpráva o stavu, hlavička protokolu a volitelně tělo, obsahující požadovaná data nebo zdroj. [47]

3.2.1 Metody

V současnosti zahrnuje HTTP devět různých metod pro různé typy akcí, zmíněny ovšem budou jen ty nepoužívanější. První je tedy metoda CONNECT, zahajující obousměrnou komunikaci, používaná hlavně u stránek komunikujících pomocí zabezpečeného HTTPS.

Další je metoda GET, kterou si klient může vyžádat nějaký specifický zdroj. Metoda by měla být použita pouze k získávání dat, neměla by sloužit ke komplexnějším úkolům, jako je vkládání nových záznamů atd. Parametry specifikující zdroj jsou viditelné přímo v adrese.

Posledními zmíněnými zde budou metody POST a PUT, sloužící pro odesílání dat na server. Data nejsou obsažena jako u metody GET v adrese, ale v těle požadavku. Rozdíl mezi metodami PUT a POST je malý, první zmíněná metoda by při opakovaném volání měla mít stále stejný výsledek, kdežto u druhé by se měl výsledek vždy lišit. [47]

3.3 SSL / TLS

Secure Socket Layer [48] je protokol, který vytváří zabezpečení komunikace pomocí kryptografických funkcí a umožňuje ověřování totožnosti komunikujících stran. Protokol vkládá „virtuální“ vrstvu mezi vrstvy transportní a aplikační v rámci modelu OSI. Pro vytvoření tohoto spojení je potřeba digitální certifikát, který jedinečně identifikuje jednotlivé účastníky komunikace. [48] Certifikáty jsou vytvářeny pomocí asymetrického šifrování, které využívá rozdílných klíčů pro šifrování a dešifrování zpráv. Certifikáty obsahují svůj zašifrovaný otisk, spolu s veřejným klíčem, pomocí kterého lze tento otisk dešifrovat a porovnat ho s námi vytvořeným otiskem přijatého certifikátu. Pokud se otisky shodují, certifikát je pravý. Zašifrovat otisk lze pouze pomocí privátního klíče, který by neměl být dostupný jiným uživatelům, než je autor certifikátu.

Nástupce tohoto protokolu, Transport Layer Security, se příliš neliší od jeho předchůdce. Jedno z vylepšení je možnost zahájení komunikace v nešifrované podobě, čímž je umožněno vytvářet zabezpečenou komunikaci se servery, které mají více než jednu doménu. [49]

3.4 WebAssembly

Jedná se o nový typ kódu, který se spouští v moderních webových prohlížečích a poskytuje nové funkce a výrazné zvýšení výkonu oproti běžně používaným technologiím. Primárně

není určen k přímému programování, ale je navržen tak, aby byl efektivním cílem kompilace pro zdrojové jazyky jako C, C++, Rust atd.

Hlavními cíli jsou především rychlost, efektivita a přenositelnost kódu. Proto kód WebAssembly lze spouštět s téměř stejnou rychlostí, jako kdyby se jednalo o nativní řešení v rámci různých platforem. I když se jedná o nízko úrovňový assembly programovací jazyk, stále je výsledný kód pro člověka čitelný, což umožňuje psaní, zobrazování i opravování kódu manuálně. WebAssembly je také zaměřen na bezpečnost, proto běží v rámci tzv. sandbox prostředí, kdy má tento vyhrazen své zdroje a hranice, které nemůže opustit, aby případně nemohl ovlivňovat ostatní programy. Technologie samotná je také zpětně kompatibilní o ostatními běžnými technologiemi. [50]

3.5 Bezdrátové přenosové technologie

Pro bezdrátový přenos informací lze využít celou řadu různých technologií a protokolů, každý s různými výhodami a použitím. Výhodou bezdrátového přenosu obecně je určitá svoboda umístění zařízení, které není nutné fyzicky zařízení připojovat k síti. Nevýhodou bývá většinou případné rušení signálu, které může posílaná data a přenos samotný zkreslovat.

Zde uvedené technologie byly během vývoje zvažovány pro implementaci komunikace mezi meteostanicí a samotným systémem.

3.5.1 Wi-Fi

Wi-Fi je bezdrátová síťová technologie, která umožňuje připojení zařízení, jako jsou počítače, mobilní telefony a další zařízení, v rámci WLAN, tedy lokální bezdrátové sítě. Využívány jsou bezlicenční pásma, nejčastěji na frekvencích 2.4 GHz a 5GHz. Pro připojení jednotlivých zařízení slouží Wi-Fi přístupový bod, mnohdy kombinovaný i s jinými zařízeními, umožňující kompatibilním zařízením připojovat se k internetu.

Protokoly používané v rámci Wi-Fi jsou definovány ve standartu IEEE 802.11. Tento standart se stále vyvíjí, přičemž se mění charakteristiky, jako je rychlost připojení, či dosah samotné sítě. S časem dochází i k vylepšování zabezpečení komunikace, která v její poslední verzi, nazývané WPA3, poskytuje ještě silnější zabezpečení oproti přechozím implementačním verzím. [51]

3.5.2 Bluetooth

Tato bezdrátová technologie slouží pro propojování zařízení, jako jsou mobilní telefony, počítače a další periferie, mezi sebou na krátké vzdálenosti. Cílem bylo nahrazení kabelů, které normálně tyto zařízení propojují, ale stále zajistit bezpečné a rychlé spojení.

Technologie samotná je méně energeticky náročná a nákladná než dříve zmíněné Wi-Fi. Cenou za tyto výhody je typicky menší dosah a menší rychlost přenosu než nejnovější standart Wi-Fi 6. Na rozdíl od Wi-Fi se navíc jedná o spojení, kdy koncové zařízení může zároveň komunikovat pouze s jedním příjemcem. Oproti tomu Wi-Fi podporuje technologii MIMO, která umožňuje přijímat signál z více zdrojů zároveň, samozřejmě v rámci jedné sítě. [52]

3.5.3 ZigBee

Protokol ZigBee byl navrhnut pro komunikaci v prostředích s velkým rušením. Klíčovou součástí jsou tzv. mesh sítě, ve kterých je každé zařízení vzájemně propojené s ostatními. Ke každému uzlu proto může existovat několik cest. Tyto sítě jsou decentralizované, proto každý jeden „node“ může nalézt „sám sebe“ v rámci sítě.

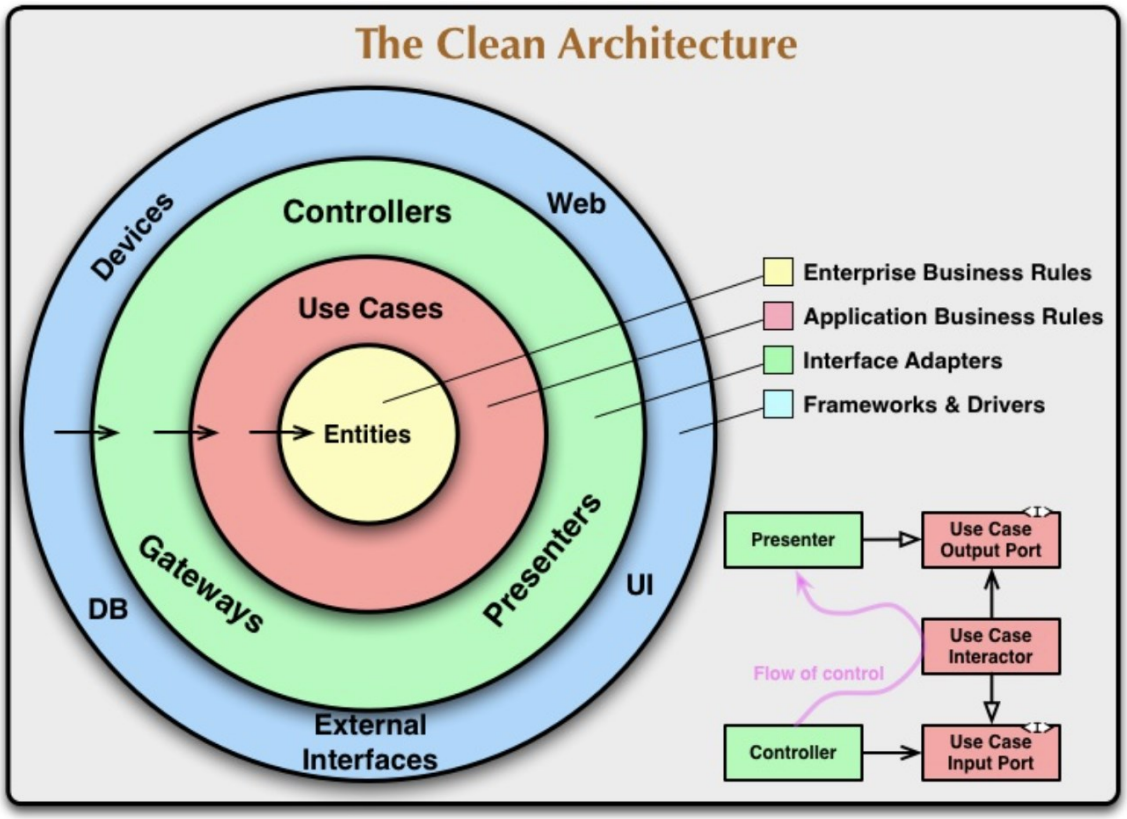
Co se použití týče, využívá se hlavně v rámci nízkonákladových a nízkoenergetických řešeních, díky čemuž některá zařízení mohou běžet z baterie po dlouhou dobu. Příkladem mohou být systémy chytré domácnosti. U těchto řešení existuje většinou pomyslné centrální zařízení, které přijímá data z koncových zařízení a následně je předává dál do podporovaných aplikací, většinou přes internet. [53]

4 NÁVRHOVÉ VZORY A METODOLOGIE

Podobný stylem, jako byly popsány některé další technologie v předchozí části, zde bude i základní popis návrhových vzorů, které byly alespoň částečně použity v rámci této práce. Je nutné poznamenat, že většina těchto vzorů nemá „pevně“ daná pravidla, a jejich interpretace je většinou na uživateli, případně společnosti.

4.1 Clean architecture

Jedná se o systém architektonických pravidel [54], vytvářející pomyslné vrstvy systému, určující rozložení jednotlivých částí systému a jejich účel.



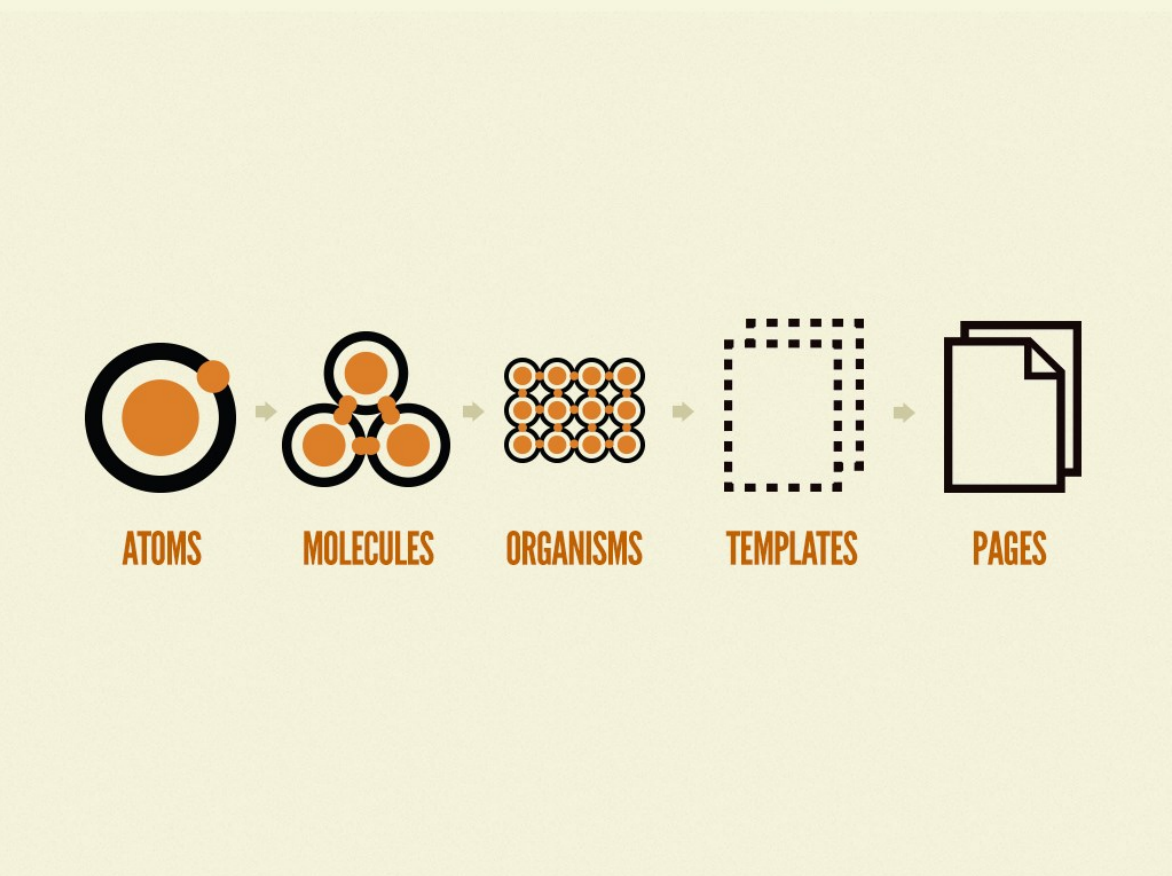
Obrázek 10 Clean Architecture [54]

Tento model sestává ze čtyř vrstev, označující různé oblasti softwaru. Vnější vrstva označuje nejnižší úroveň z hlediska funkcionality systému, přičemž postupem do středu se tato úroveň zvedá. Každá další vrstva by se také měla co nejméně ve výsledném řešení měnit, aby byla zachována základní funkcionalita a podpora těchto funkcí. Vrstvy by měly být od sebe také odděleny přístupově, tedy vnější vrstvy mohou přistupovat ke zdrojům vnitřních vrstev, ovšem vnitřní vrstvy by neměly vidět a moci přistupovat k vrstvám vnějším.

První vrstva, brána z vnějšku, obsahuje pomyslný základ systému. Zahrnuje různé funkce, přístup k databázi, externí rozhraní, jako jsou platformní API apod. Další vrstva obsahuje hlavně komponenty s logikou aplikace, především různé kontrolery pro přístup ke zdrojům. Třetí vrstvou je aplikační byznysová logika, poskytující případy užití. Jednotlivé moduly jsou zde také koordinovány a konkrétně používány. Poslední vrstvou je vrstva podniková byznysová, která obsahuje hlavně entity, nebo také objekty, definující strukturu aplikace. [54]

4.2 Atomic design

Atomický design [55] je metodologie určená na tvorbu designových systémů, jako jsou například webové stránky. Hlavním principem, který je inspirován přírodními prvky, je rozdělení celku na součásti, které lze následně skládat dohromady a tím tvořit komplikovanější struktury.



Obrázek 11 Atomic Design [55]

Celkem je pět druhů prvků. Atomy, Molekuly, Organismy, Šablony a nakonec Stránky. Atomy slouží jako základní stavební bloky. Tvoří jednoduché prvky, jako je text, tlačítka

nebo různé vstupní prvky. Mohou obsahovat také abstraktní prvky, jako jsou barevné palety nebo fonty.

Z atomů se vytvářejí molekuly. To jsou skupiny atomů sdružené za nějakým účelem, například pro definování vyhledávací pole, které kombinuje atomy pro textový vstup a tlačítko.

Dalším prvkem jsou organismy, které opět kombinují předchozí prvky do prvků s komplexnější funkcionalitou. Zde jsou už definovány některé části výsledné stránky, například hlavička stránky, menu či navigace v rámci stránky.

Šablonami následně vzniká rozložení samotné stránky, obsahující jednotlivé prvky složené z přechozích prvků. Implementují finální vzhled, který ovšem postrádá detaily a data samotná. Ty jsou přidány v poslední části atomického designu, ve Stránkách.

Výhodou tohoto systému je hlavně modularita a schopnost vytvářet nové prvky za použití skládání stávajících objektů dohromady. Změnami v nižších vrstvách lze poté upravovat celkový vzhled stránek, bez nutnosti přepisovat více objektů a možné nekonzistenci kvůli tomuto upravování. [55]

4.3 Dependency Injection

Návrhový vzor Dependency Injection je používaný na implementaci tzv. Inversion of Control, kdy místo toho, aby se třídy přímo staraly o vytváření instancí jiných tříd, se toto děje v rámci „vyšších“ funkcí. Instance tříd jsou tedy vytvořeny mimo třídy samotné a následně poskytnuty jako reference. DI je tedy specifická varianta či implementace IoC.

Návrhový vzor využívá tři druhy tříd. Klientskou třídu, která vyžaduje instanci jiné třídy, třídu se službou samotnou, a nakonec tu třídu, která se stará o ono vkládání závislostí.

Možností, jak s DI, je několik. Jednou je již zmíněné vkládání do konstruktoru klientské třídy, ve kterém je třída uložena do lokální proměnné, ke které se poté přistupuje. Další je možnost vkládání závislostí přes veřejnou vlastnost třídy, taktéž nazývaný jako Setter Injection a poslední je vkládání metod, kdy obsah tříd je definována pomocí rozhraní, přičemž implementace obsahu metod je určena právě ve třídě starající se o DI. [56]

Implementace a podpora se liší od programovacího jazyku, uvedené příklady proto nemusí fungovat u všech jazyků. Všechny by ovšem měly fungovat v rámci jazyku C#, který je použit ke tvorbě této práce.

4.4 MVC

Základní motivací, která vedla k vytvoření architektonického vzoru Model-View-Controller, bylo oddělení logiky aplikace od zobrazení výstupů. Rozděluje interní logiku aplikace podle druhu a zaměření kódu, čímž zpřehledňuje kód v projektu a zlehčuje jeho úpravy a údržbu.

Celkem tento vzor obsahuje tři typy komponentů. To jsou modely, které obsahují logiku aplikace, jako jsou například různé výpočty, dotazy, validace atd. Model samotný by neměl znát zdroj těchto dat nebo kam budou po zpracování odeslána, ani jak budou vypadat. Další částí tohoto vzoru jsou pohledy, které definují vzhled výstupních dat a jak se prezentují uživateli. Podobně, jako u modelů je nezajímá zdroj dat a nestarají se o vnitřní logiku aplikace. Poslední částí je kontroler, sloužící jako prostředník mezi modely a pohledy. Předává dotazy z pohledů na modely a výsledná data z modelů na pohledy. Data jsou většinou upravena do podoby, která vyhovuje dané části. [57]

4.5 REST API

Restful API je aplikační programové rozhraní, které odpovídá požadavkům REST architektury a umožňuje interakci s RESTful webovými službami. Pod zkratkou REST se skrývá název Representational State Transfer a byl vytvořen Royem Fieldingem. API je zase zkratka pro Application Programming Interface. [58]

4.5.1 API

Jedná se o sadu definicí a protokolů pro vytváření a integrování komunikace mezi aplikacemi. Občas je na něj odkazováno jako na kontrakt mezi poskytovatelem informací a příjemcem informací, kdy jsou definovány požadavky pro obě strany. Taktéž lze na API pohlížet jako na prostředníka mezi klientem a serverem. [58]

V dnešní době jsou často API považovány za produkt než za kód. Jsou navrženy, aby byly konzumovány specifickým publikem, mají svou dokumentaci a také jsou často verzovány. Také mají svůj cyklus vývoje, jako klasický software, zahrnující testování, sestavování, spravování a verzování. [59]

4.5.2 REST

REST je sada architektonických pravidel, nejedná se tedy přímo o protokol nebo standart, přičemž implementace v rámci API záleží na vývojáři samotném. Je populární hlavně kvůli své jednoduchosti a faktu, že se postaven na již existujících systémech protokolu HTTP. To

také určuje jeho hlavní výhody, jako je komunikace za využití HTTP kódů, a jeho založení na zdrojích, které taktéž vychází z protokolu HTTP. Také je nezávislý na konkrétním jazyku a velmi rozšířený jak na straně klienta, tak i na serverech. [60]

5 METEOROLOGIE

Meteorologie je věda, zabývající se studiem atmosférických fenoménů, převážně v troposféře a nižní stratosféře. Zahrnuje také systematické studium počasí a jeho příčin, přičemž poskytuje základ pro předpovědi počasí jako takového. [61]

5.1 Historie

Počátky má meteorologie ve starověkém Řecku, kde procesech. V tomto období vytvořil řecký filozof Aristoteles, na základě ranných pozorování a teorií o atmosférických jevech, první souhrnný přehled o těchto atmosférických vědách, zvaný *Metetorologica.* Tyto teorie zůstaly z většiny nepopřeny až do příchodu meteorologických zařízení a nových teorií, které vycházely z více objektivních a četnějších dat a pozorování.

Přesto, že první srážkoměr byl vyvinut v Indii už kolem roku 400 př. n. l., většina důležitých instrumentů byla vynalezena až kolem 16. a 17. století. V roce 1450 byl vyroben mechanický anemometr, tedy zařízení pro měření intenzity větru. Co se měření teploty týče, první takového zařízení vynalezl Galileo Galilei v roce 1592. První jednotné stupnice na měření teploty ovšem nebyly vyvinuty až do 18. století, kdy vznikly stupnice pro stupně Fahrenheitů a stupňů Celsia.

Další klíčový pokrok přišel v období druhé světové války, kdy se technologický pokrok dostal do popředí, a také vynálezem radaru v roce 1935. V roce 1950 John von Neumann a jeho kolegové dokázaly za pomoci prvních matematických atmosférických modelů vytvořit první počítačem generovanou předpověď počasí. Následný vynález satelitů určených na pozorování počasí, Dopplerova radaru a dalších pokroků formovalo vědu o počasí až podoby, v jaké ji známe dnes. [62]

5.2 Meteorologické veličiny

Přístroje poskytly kvantitativní a objektivní prostředek, jak popisovat současný stav okolního prostředí. Přestože kvalitativní popisy jsou stále běžné, kvantitativní měření a statistické souhrny prvků počasí jsou dnes běžně používány v televizi, rádiu, na internetu, v novinách atd. Mezi obecně měřené a prezentované veličiny počasí patří proměnné, jako je teplota, vlhkost, atmosférický tlak, pohyb větru a oblačnost. [62]

5.2.1 Teplota

Teplota vzduchu označuje stupeň „horkosti“ nebo „chladu“ atmosféry. Z fyzikálního hlediska je teplota průměrnou kinetickou hodnotou energie molekul. Rychleji pohybující se molekuly mají vyšší teplotu než ty, co se pohybují pomalu.

Standartně se teplota uvádí ve stupních Celsia, speciální případy používají stupně Fahrenheita. Pro vědecké účely se běžně používají Kelviny, které mají stejný rozsah mezi hodnotami jako u stupňů Celsia, pouze posunuté o přibližně 273 jednotek.

Hodnoty se mohou měřit jako maximum, minimum či průměr během nějaké doby, například hodin, dnů a až roků. Rozdíly v těchto měřeních jsou poté používány k určení stability atmosféry a dalších parametrů. [62]

5.2.2 Vlhkost

Různé proměnné existují také pro měření vlhkosti, také označovanou jako množství vody v atmosféře. Nejpoužívanější v rámci předpovědí je RH, tedy poměr množství vodní páry v atmosféře ku množství potřebnému k nasycení atmosféry za současných teplotních a tlakových podmínek. Hodnota samotná je udávána v procentech.

Denní teplotní cyklus má tendenci podporovat vyšší procento vlhkosti během ranních hodin, kdy teplota vzduchu je chladnější a nasycení vodou je pravděpodobnější než v odpolední hodiny, kdy se jedná o pravý opak. [62]

5.2.3 Tlak

Atmosférický tlak je množství síly, kterou působí atmosféra planety na určitou plochu. Mezinárodní systém měření (SI) definuje atmosférický tlak jako Newton na metr čtvereční (N m^2) nebo Pascal (Pa).

Tento tlak je také ovlivněn nadmořskou výškou, protože s vyšší výškou tlak klesá a naopak s nižší výškou stoupá. Proto bývá základní hodnota tlaku přizpůsobena tlaku v nulové výšce nad mořem, kde je průměrná hodnota 101 100 Pa.

Pomocí tlaku lze do určité míry předpovídat počasí. Vysoký tlak většinou značí bezoblačnost a klidné podmínky, kdežto nízký může předvídat například bouřku. Tlak také definuje pohyb mraků, směr větru a další proměnné. [62]

5.2.4 Vítr

Vítr je popisován parametry, jako je směr, rychlost a poryvy, přičemž jsou jako takové měřeny vzhledem k rotaci Země. Směr je většinou určen podle světových stran pomocí stupňů rotace. Rychlost větru je zase změna vzdálenosti v čase nebo relativní rychlost v rámci místní atmosféry, měřená většinou v kilometrech za hodinu. [62]

5.2.5 Další veličiny

Existuje celá řada dalších parametrů, které lze měřit a zaznamenávat. Může se jednat o intenzitu slunečního svitu nebo jeho UV záření, případně i veličiny počítané, například rosný bod.

5.3 Meteorologická stanice

Meteorologická stanice je soubor přístrojů, které měří atmosférické podmínky a pomáhají tak studovat počasí a klima konkrétního místa. Většina meteorologických stanic měří běžné veličiny, jako teplotu, vlhkost, barometrický tlak, rychlost a směr větru nebo srážky.

Profesionální meteorologické stanice se používají pro letecké a vojenské použití, četně využití pro předpověď počasí a jeho výzkum, zatímco osobní domácí meteorologické stanice jsou většinou určeny pro soukromé nebo v některých případech pro výzkumné použití. Nejčastějšími vlastníky osobních meteostanic jsou meteorologové, farmáři a také školy. [63]

5.3.1 Profesionální

Tyto zařízení bývají děleny na několik typů. Poloautomatické, u kterých je potřeba manuálně zapisovat naměřené hodnoty, a automatizované, které data odesílají sami. Také se mohou lišit v komplexnosti, zaměření a také poloze.

Data sestavená oficiálními stanicemi využívají meteorologové jako pomoc při vytváření předpovědí a sdílejí je prostřednictvím internetu a dalších zdrojů. [63]

5.3.2 Osobní

Osobní meteorologické stanice nejsou zdaleka tak složité a nemají takovou přesnost jako ty profesionální. Dražší modely jsou však dostatečně schopné pro použití i ve výzkumných aplikacích. Osobní stanice jsou často připojeny k internetu, aby sdílely data a monitorovaly podmínky na dálku prostřednictvím různých platforem a systémů. Jejich měření mohou být také využita v rámci integrace s chytrou domácností. [63]

II. PRAKTICKÁ ČÁST

6 NÁVRH

6.1 Motivace

Jedním z hlavních důvodů, proč vůbec tento nápad na aplikaci / meteostanici vzniknul, byla moje babička. Ta si už pár let zapisuje, jaké bylo, v který den počasí, a pak mé rodině vykládá, jaké bylo v minulých letech v daný den abnormální počasí. Třeba že v půlce dubna sněžilo, v prosinci bylo 20 stupňů a podobně. To mě motivovalo k tomu, že bych mohl vytvořit systém, který by tato data uchovával a pak každý rok mohl ukazovat tyto historické hodnoty. Dokonce by se mohl i pokoušet „předvídat“ počasí, což by byl úkol nelehký a produkoval by nejspíše data velice nepřesná. Jako přibližný odhad by ovšem posloužit mohl.

Dalším z důvodů je kvalita komerčních meteostanic. Jak je popsáno dále, existuje řada „profesionálních“ meteostanic, které ovšem nesplňují mé představy a požadavky. Proto jsem se rozhodl vytvořit vlastní řešení, které mi bude vyhovovat a při jehož tvorbě se naučím spoustu nových a užitečných znalostí.

Co se týká internetových služeb, které poskytují meteorologická data a předpovědi, jedná se o velmi kvalitní zdroje využívající profesionálních zaznamenávacích zařízení, ovšem u nich nastává problém, že nejbližší meteostanice, která do jejich systému přispívá, je často velmi vzdálená od lokality, která nás jako uživatele zajímá a data nemusí i přes jejich kvalitu odpovídat lokálním podmínkám.

6.2 Existující řešení

Na trhu existuje spousta jak komerčních, tak DIY projektů zaměřených na sledování a zaznamenávání počasí. Může se jednat o obyčejné „hloupé“ teploměry, které ukazují v podstatě jenom aktuální teplotu, čas a občas i základní předpověď, nebo přímo o celou meteostanici, která ovšem ve většině případů používá pro ukládání dat a komunikaci nějaký druh proprietární technologie, čímž jakákoliv možnost rozšiřování funckionalit nepřipadá v úvahu.

Co se týká projektů zdarma dostupných řešení, tak ty sice jsou většinou zdarma a případně open source, ovšem podobně jako u komerčních řešení používají k ukládání nebo získávání dat nějakou existující službu, která buď opět neodpovídá požadavkům, nebo je placená. Pokud externí službu nepoužívá, jedná se zpravidla pouze o koncové zařízení bez možnosti

ukládání dat, nebo o uzavřený software, který sice má pár užitečných funkcí, ale zase mu spousta funkcionalit chybí.

6.2.1 Software

6.2.1.1 WeeWX

Jako první příklad „konkurenčního“ softwaru lze uvést WeeWX [64]. Jedná se o open source program, napsaný v jazyce Python, který interaguje s meteostanicemi a vytváří grafy, přehledy a HTML stránky. Jako klíčové prvky uvádí podporu celé řady meteostanic, možnost nahrávat data na webové servery, jednoduchost instalace a další. [64]

Oproti řešení vytvořené v rámci této práce se jedná o robustnější projekt s dlouhým vývojem za sebou, umožňující rozsáhle možnosti přizpůsobení a dalších funkcí. Náš systém ovšem sází více na jednoduchost a uživatelskou přívětivost, vycházející hlavně z uživatelského rozhraní, které u systému WeeWX je sice přizpůsobitelné, nejedná se ovšem většinou o nijak moderní či lákavé prostředí. Náš systém je také zaměřen na jiný typ sběru dat a dat samotných, které jsou prostší a nejsou poskytovány systému nonstop, pouze v daných intervalech.

6.2.1.2 OpenWeatherMap

I když se nejedná o software, který by se staral přímo o meteostanice a sběr dat z nich, je vhodné OpenWeather [65] zmínit. Jedná se o službu hlavně určenou na předpověď počasí, která navíc umožňuje pomocí vlastního API získávat aktuální data ze zvolené lokality.

Služba poskytuje řadu rozdílných dat, některá ovšem přístupná pouze v rámci jedné z variant předplatného. O základní verze, která je přístupná zdarma po registraci do systému a stále poskytuje spoustu relevantních dat, až po placenou podnikovou s funkcemi, jako jsou historické mapy, globální předpovědi atd. [65]

Když tento systém porovnáme s naším řešením, tak zaměřením jsou to dva poměrně odlišné systémy. OpenWeather je zaměřen na globální měřítko a poskytuje data i ze zahraničí, kdežto náš systém je orientován lokálně a sbírá data, která jsou více relevantní pro uživatele, když se pohybuje v blízkosti umístění meteostanice. Sice zahrnuje i možnost více zařízení a domácností, ovšem stále se bavíme o poměrné malé lokalitě, případně „ostrovy“ pokrytí.

I přes rozdílnost se ovšem nejedná přímo o konkurenci. Jako jedna z variant, které jsou zmíněny dále, je možnost systém rozšířit o získávaní dat z externích zdrojů pomocí služeb

stejného typu, jako OpenWeather, četně. Konkrétně by se mohlo jednat o funkci zjištění aktuálního počasí či přímo předpovědi pro oblast, kterou uživatel plánuje navštívit.

6.2.1.3 ThingSpeak

ThingSpeak [66] je IoT analytická platforma, která umožňuje agregovat, zobrazovat a analyzovat živá data v cloudu. Data lze posílat z různých zařízení, z nich vytvořit vizualizace a také použít tato data jako upozornění odesílaná pomocí služeb, jako je třeba Twitter. Pro zpracovávání dat je použit software MATLAB, díky kterému může služba vytvářet kód určený pro předzpracování, vizualizaci a analýzy našich dat. Kromě omezené verze zdarma poskytuje služba možnost platby za další funkce, počet zařízení, podporu a další. [66]

V porovnání s naším řešením je tato služba velmi rozsáhlá a zaměřená i na jiné typy dat a projektů. Funkcionálně obsahuje hlavně nástroje, které využívají platformu MATLAB, čímž poskytuje i možnosti přesahující rozsah naší práce. Toto zaměřením ji také činí ne příliš přívětivou pro běžného uživatele, co se vizualizací a práce se systémem týče.

Podobně jako u služby OpenWeatherMap, tak i zde je potenciál na propojení s naším systémem, hlavně za účelem analýzy nasbíraných dat, která může být velmi kvalitní s ohledem na tuto službu.

6.2.2 Hardware

6.2.2.1 Weather Station V3.0

Tento DIY projekt [67] je zaměřen na samotnou meteostanici a její hardware. Cílem je zde vytvořit solárně napájenou meteostanici, která komunikuje pomocí Wi-Fi. Jedná se už o třetí iteraci tohoto projektu, kdy každá další verze přidává další funkcionalitu a částečně mění implementaci. Celý projekt je velmi podrobný tomu v této práci, co se týká instrukcí a jednotlivých designových rozhodnutí. [67]

Jako výhody tohoto projektu lze pokládat hlavně celková doba existence a jeho iterace, čímž byly vyladěni některé neduhy. Taktéž projekt využívá na zakázku vyrobené „profesionální“ desky plošných spojů, které propojují jednotlivé komponenty meteostanice. Tím je poskytnuta větší odolnost meteostanice a také projekt jako takový působí elegantnějším dojmem. Oproti systému v této práci ovšem tento projekt nemá vlastní systém, na který by odesílal data a který by s ním dokázal spolupracovat. Místo toho využívá k ukládání dat výše zmíněný ThinkSpeak.

Z tohoto projektu také částečně vychází i meteostanice vytvořená v této práci. Myšlenka na vytvoření tohoto systému vznikla dlouho před uveřejněním tohoto projektu, ovšem díky shodě základního konceptu a hardwaru byl využit k vylepšení naší verze. Hlavním rozdílem je jiný mikrokontroler kontrolující meteostanici, bytelnější konstrukce vytvořená z hliníkových tyčí a také způsob napájení, vycházející právě z jiné volby hlavního ovládacího prvku.

6.2.2.2 Open Weather Station

Dalším příkladem hardwaru je zde uveden tento DIY projekt [68], zaměřený na dostupnou, stabilní a jednoduše sestavitelnou meteostanici. Vyvíjená je od roku 2012 a kromě již zmíněných zaměření je dalšími výhodami kompaktnost, možnost využití Wi-Fi a mobilní sítě, ochrana proti ztrátě napájení.

Konceptuálně je také podobný naší a předchozí zmíněné meteostanici. S předchozím řešením je zde také podobnost, co se týče meteorologických dat, jelikož taktéž spoléhá na externí službu, jako je TeamSpeak nebo OpenWeather. Data jsou v tomto případě předávána přes aplikaci určenou pro mobilní zařízení se systémem Android, čímž vzniká jakýsi mezikrok v předávání dat. V dalších aspektech jsou ovšem tyto systémy celkem odlišné. V3 využívá jako mikrokontroler ESP32, který již má integrovaný modul pro komunikaci pomocí Wi-Fi, kdežto tato stanice používá jako hlavní kontrolér Arduino UNO a ke komunikaci Bluetooth modul. Ve zdroji napájení jsou oba projekty také rozdílné. První je navrhnutý pro solární napájení, ale druhý, i když má tuto možnost, je primárně napájen externím zdrojem. V neposlední řadě lze uvést rozdíl v provedení desky plošných spojů, která v případě tohoto projektu je vytvářená „po domácku“, oproti profesionálnějšímu provedení prvního projektu. [68]

V porovnání s naší verzí zde vzniká nevýhoda v nutnosti využívat Android aplikaci pro komunikaci s meteostanicí. Data lze mít v tomto případě uložena pouze po dobu 60 dní, čímž odpadá přímá využitelnost pro archivaci dat bez nutnosti externí služby. Jako nevýhodu lze také považovat použití Arduino UNO, které je oproti alternativám náročnější na energii, čímž není toto řešení tak vhodné pro solárně napájené řešení.

Podobně jako u meteostanice V3, i tato varianta by byla využitelná jako inspirace při implementaci meteostanice v této práci. Blíž má ale naše zařízení spíše k prvnímu řečenému, hlavně kvůli větší celkové podobnosti s naším projektem.

6.3 Návrh

Základní funkcionalitou tohoto systémů je zaznamenávání dat, která se musí dát prezentovat v nějaké formě uživateli. Systém musí také umožňovat data ukládat a do jisté míry i upravovat, přidávat a spravovat nové zdroje dat, a v neposlední řadě kontrolovat, kdo má ke konkrétním datům a nastavením přístup.

Jako nejvhodnější implementace řešení byla zvolena webová aplikace. Ta se bude skládat z klientské části, která bude umožňovat zobrazování dat a nastavování aplikace v podobě, která je uživatelsky přívětivá, a serverové části, která se bude starat o přijímání, ukládání a zobrazování dat, o autentizaci uživatelů a o celkovou funkčnost systému.

Serverová část bude umožňovat komunikaci jak s klientskou částí, tak se zdroji dat, a to pomocí tzv. RESTful API. To poskytuje přístupové body, ke kterým lze přistupovat a podle druhu daného přístupu data číst, zapisovat a upravovat je, případně i mazat. Taktéž se jedná o hlavní bod zabezpečení celé aplikace, kdy pro přístup k těmto bodům je třeba speciální autentizační token, potvrzující identitu a oprávněnost uživatele přistupovat k těmto zdrojům.

Jako způsob získávání a vkládání dat do systému samotného byla zvolena meteostanice, která komunikuje pomocí HTTP(S) protokolu přes internet, což má řadu výhod. Oproti alternativním řešením, jako je Bluetooth nebo ZigBee, které mají buď vyšší dosah, menší spotřebu nebo jinou kladnou vlastnost, je výhodou použití internetového protokolu jeho přístupnost v podstatě odkudkoliv. V blízkosti bydliště uživatele může používat Wi-Fi, případně i drátové připojení k síti, přes které lze případně meteostanici i napájet, nebo mobilní sítě, jejichž pokrytí je v dnešní době velmi rozsáhlé a pro posílání pár kilobitů dat i dostatečně rychlé.

Díky těmto volbám je zde patrná výhoda, že jako zdroj dat lze využít jakékoliv jiné zařízení, které umí komunikovat přes internet. Meteostanice může komunikovat i jiným způsobem a přijímač, pokud je možné ho připojit k internetu, může tyto data upravit do vhodného tvaru a poslat, jako by se jednalo o meteostanici samotnou. Taktéž se nemusí nutně jednat o nějaké hardwarové koncové zařízení, které data vysílá. Může se jednat i o webovou službu, která je zaměřena na sběr meteorologických dat a využít ji jako zdroj pro náš systém.

6.3.1 Funkční a nefunkční požadavky

V této sekci jsou definovány požadavky na systém jako takový. Zahrnuje to požadovanou funkcionalitu systém z pohledu uživatele a parametry systému samotného. Tyto požadavky

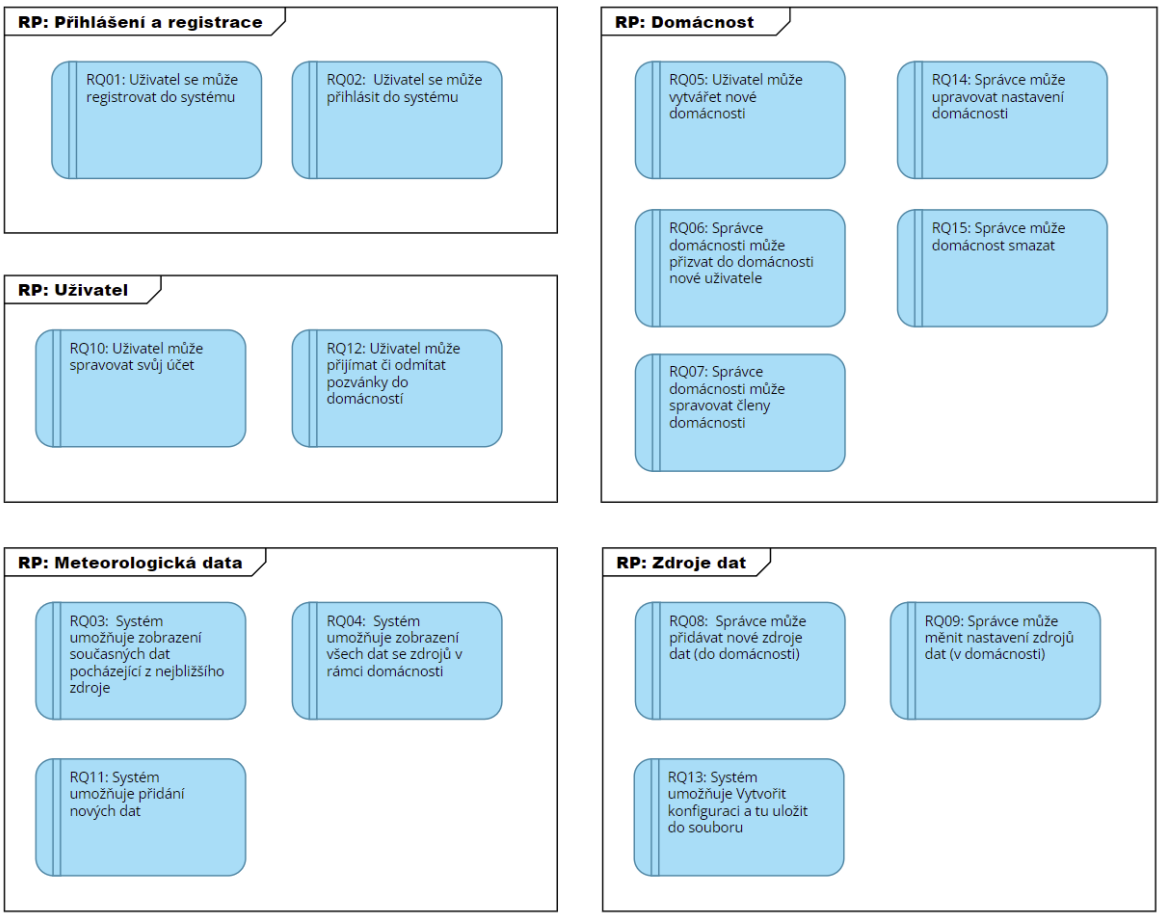
následně musí být pokryty alespoň jedním případem použití, jak je zobrazeno dále v sekci pokrytí požadavků.

Zjednodušeně řečeno, funkční požadavky určují funkce systému, se kterými bude uživatel přímo interagovat. Zahrnují funkce jak základní, jako je přihlášení nebo registrace do systému, tak pokročilejší a konkrétnější, jako je zobrazování požadovaných dat nebo přidávání nových záznamů.

Oproti funkčním požadavkům se ty nefunkční zaměřují na funkcionalitu a parametry systému z techničtějšího prohledu. To jsou hlavně specifika, jako je dostupnost systému, jeho zabezpečení nebo robustnost či rozšiřitelnost.

6.3.1.1 Funkční požadavky

Požadavky jsou zde rozděleny do několika částí, podle jejich zaměření a účelu. Prvními jsou požadavky na přihlášení a registraci do systému. Smyslem je poskytnout uživateli možnost vytvořit si v rámci systému účet, a moci se s jeho pomocí do systému opakovaně přihlásit.



Obrázek 12 Funkční požadavky

Další je kategorie Uživatel, obsahující požadavky umožňující uživateli nastavovat svůj účet a možnost přijímat a odmítat pozvánky do různých domácností. Předvolby, které si uživatel zvolil, zůstanou stejné a nepřístupné ze strany jiného uživatele.

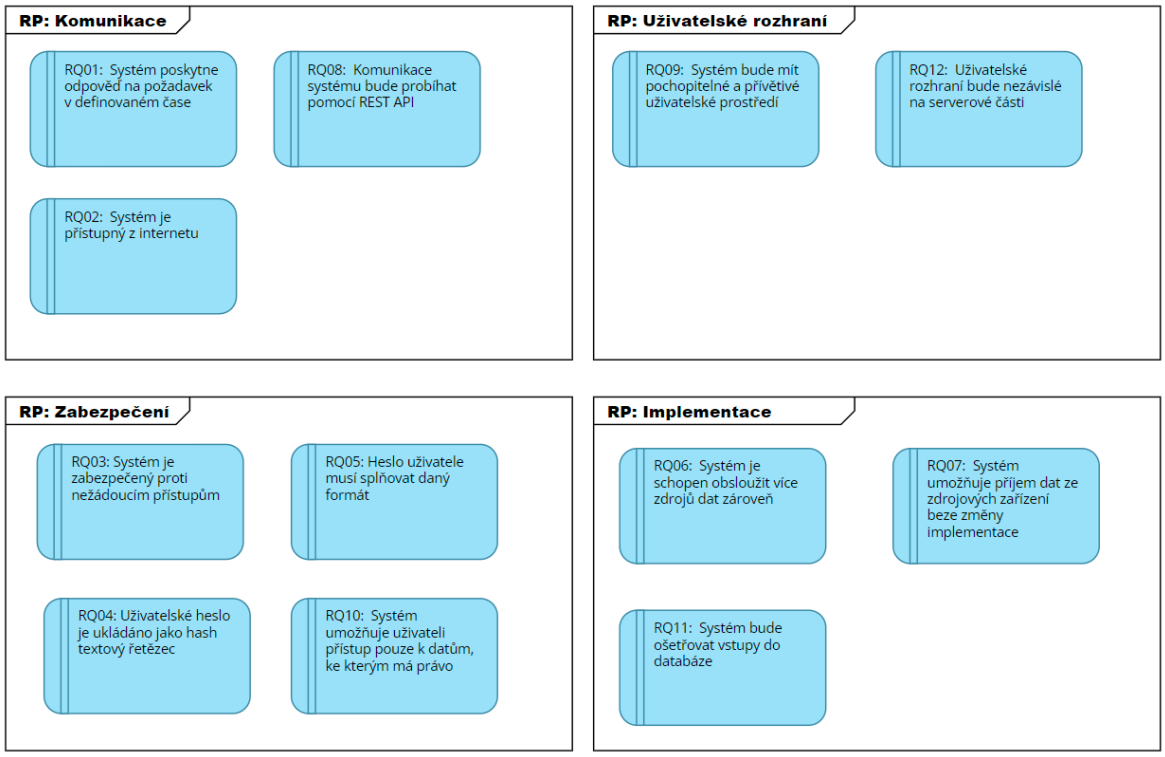
Třetí sérií požadavků jsou zahrnuty v sekci Meteorologická data. Požadavky jsou zde zaměřeny na možnosti zobrazení zvlášť aktuálních a historických meteorologických dat v dané domácnosti. Nakonec je zde definována možnost přidávat nové meteorologické záznamy do systému.

Následuje sekce Domácnost, zaměřená na stejnojmennou funkcionalitu. Systém musí podle této sekce obsahovat funkce pro vytváření nových domácností a mít možnosti správy uživatelů, čímž je myšleno přizvání uživatelů, jejich následná správa, a možnost je z domácnosti i odebrat. Taktéž musí systém umožňovat spravovat domácnost jako takovou a umožnit ji ze systému smazat. S domácnostmi je spojena i většina ostatních požadavků, i když ne přímo.

V posledních požadavcích, zahrnutých v sekci Zdroje dat, jsou určeny funkce pro přidávání nových zdrojů dat, tedy meteostanice, možnost měnit jejich nastavení a také možnost tyto zdroje přímo konfigurovat. Tím je myšleno vyplnění formuláře na straně klienta, přičemž poskytnuté údaje se nahrají do šablony kódu pro meteostanici. Takto vyplněný soubor se potom uloží na klientském zařízení pro následné nahrání do meteostanice samotné.

6.3.1.2 Nefunkční požadavky

Podobně jako u funkčních požadavků, jsou i ty nefunkční rozděleny do sekcí podle jejich zaměření. První z nich je Komunikace, popisující, jak by měl systém komunikovat s ostatními částmi systému a s uživatelem. Hlavními částmi je požadavek na odpověď systému v definovaném čase, požadavek na přístupnost systému z internetu a požadavek na komunikaci, konkrétně pomocí REST API.



Obrázek 13 Nefunkční požadavky

Další sekcí je uživatelské rozhraní, které by podle stanovených požadavků mělo byt pochopitelné a přívětivé pro koncového uživatele. Také by mělo být uživatelské rozhraní, tedy klientská část aplikace, nezávislá na serverové části. To znamená, že implementace klientské části by neměla být nedílnou součástí celku a měla být nahraditelná, například mobilní aplikací.

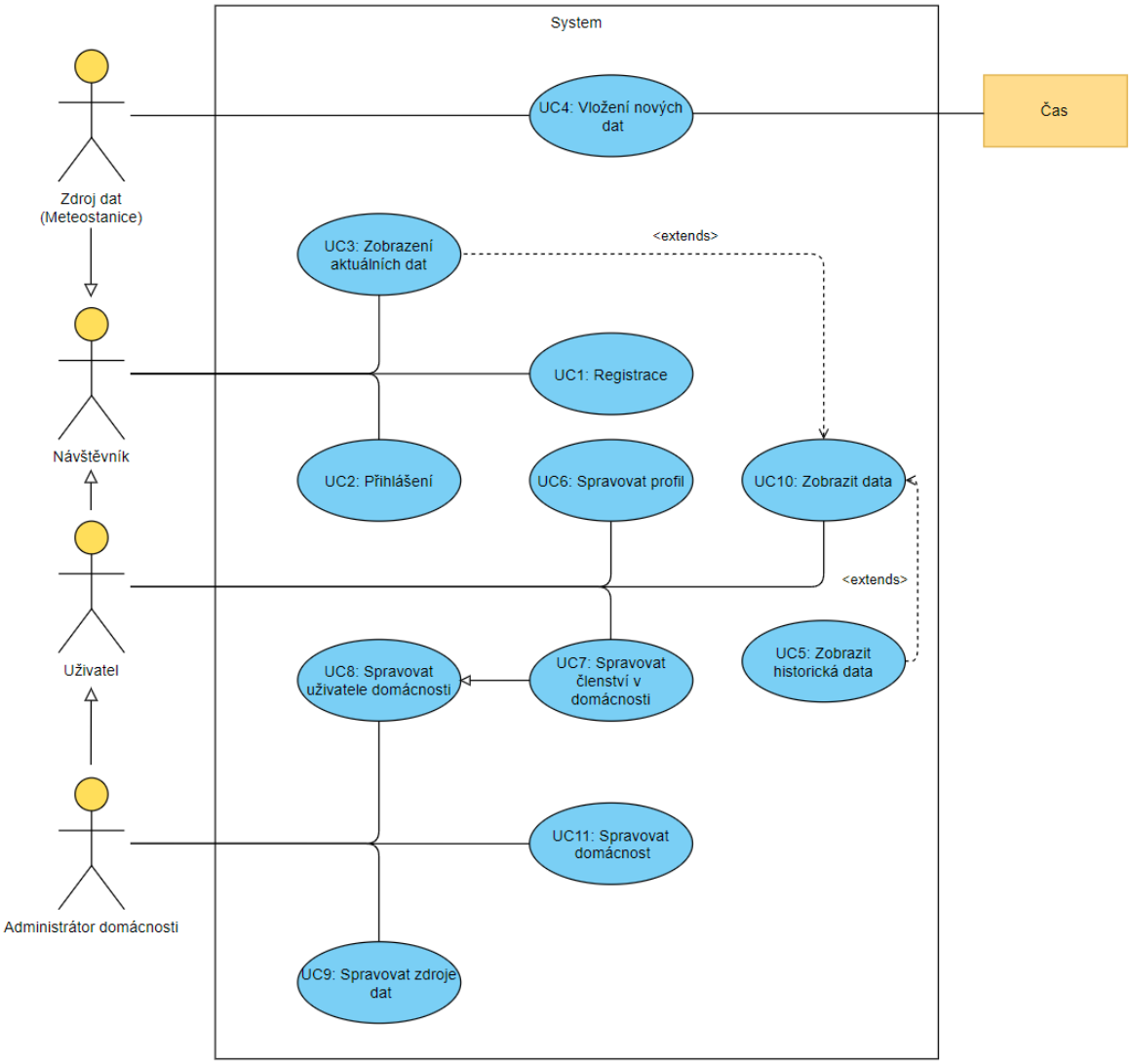
Nedílnou součástí systému je také zabezpečení, s požadavky popsanými ve stejnojmenné sekci. Ta obsahuje požadavky na zabránění nežádoucích přístupů do aplikace, zamezení přístupu k datům, ke kterým nemá uživatel přístup a požadavky na hesla. Ty se týkají hlavně způsobu uložení hesla v bezpečné formě, tedy jako HASH otisk, a formátu samotného hesla, potažmo požadavků na znaky a jejich počet, které by mělo heslo obsahovat.

Jako poslední sekce je zde uvedena Implementace, která se zaměřuje na možnost obsloužit více meteostanic zároveň, možnost přijímat data z různých zdrojů s různou implementací, a nakonec i ošetření vstupních dat jako takových.

6.3.2 Případy užití

Případy užití slouží k popsání kroků nebo akcí, které slouží k definování interakce mezi rolemi a částmi systému. Také pomocí nich lze popsat, jak uživatel provádí nějakou akci a co

během ní v systému děje. Každý případ popisuje posloupnost akcí, které musí uživatel provést k tomu, aby provedl akci popsanou konkrétním požadavkem. To může zahrnovat kliknutí na položku v menu, vyplnění nějakého pole, plus následnou reakci ze strany systému. [69]



Obrázek 14 Případy užití

6.3.2.1 Konkrétní příklad

Pro příklad zde bude uveden jeden konkrétní případ užití, konkrétně ten týkající se registrace. Podobným způsobem je možné popsat i ostatní případy.

Případ začíná kliknutím uživatele na ikonu registrace na úvodní obrazovce aplikace. Zde zadá své informace, konkrétně uživatelské jméno a následně heslo, pomocí kterých se bude přihlašovat. Heslo musí obsahovat minimálně jedno velké písmeno, jednom malé, dále

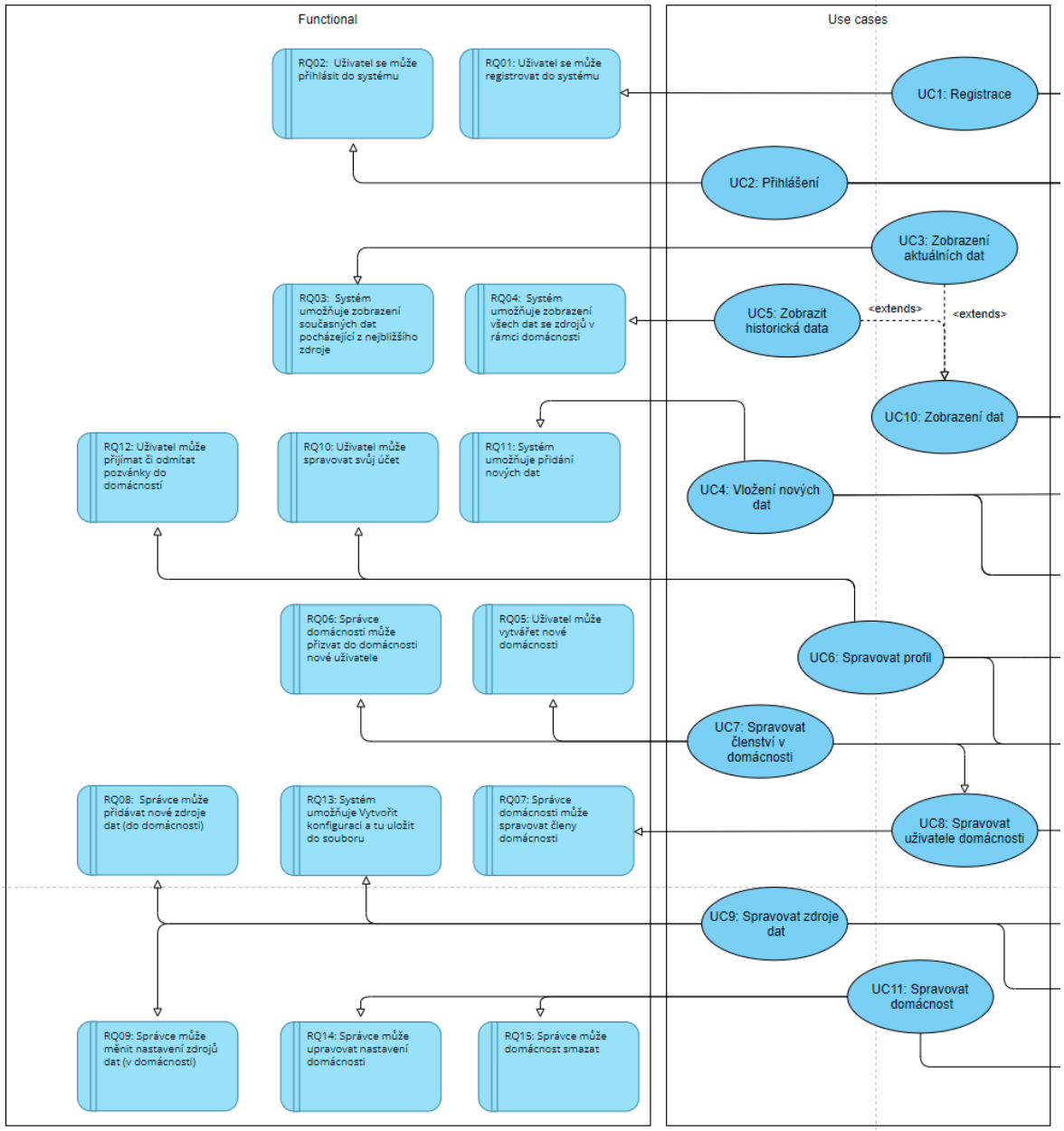
speciální znak a musí být minimálně šest znaků dlouhé. Po zadání údajů uživatel potvrdí registraci kliknutím na tlačítko registrovat. Na základě zadaných údajů je vytvořen požadavek, který je odeslán na serverovou část aplikace, kde je zpracován. Po přijatí tohoto požadavku je provedena opětovná kontrola hesla a ověření, že se už v systému nenachází uživatel se stejným jménem. Pokud je kontrola úspěšná, je vytvořen nový uživatel a na klientskou část odesláno potvrzení o úspěšné registraci.

6.3.3 Pokrytí požadavků

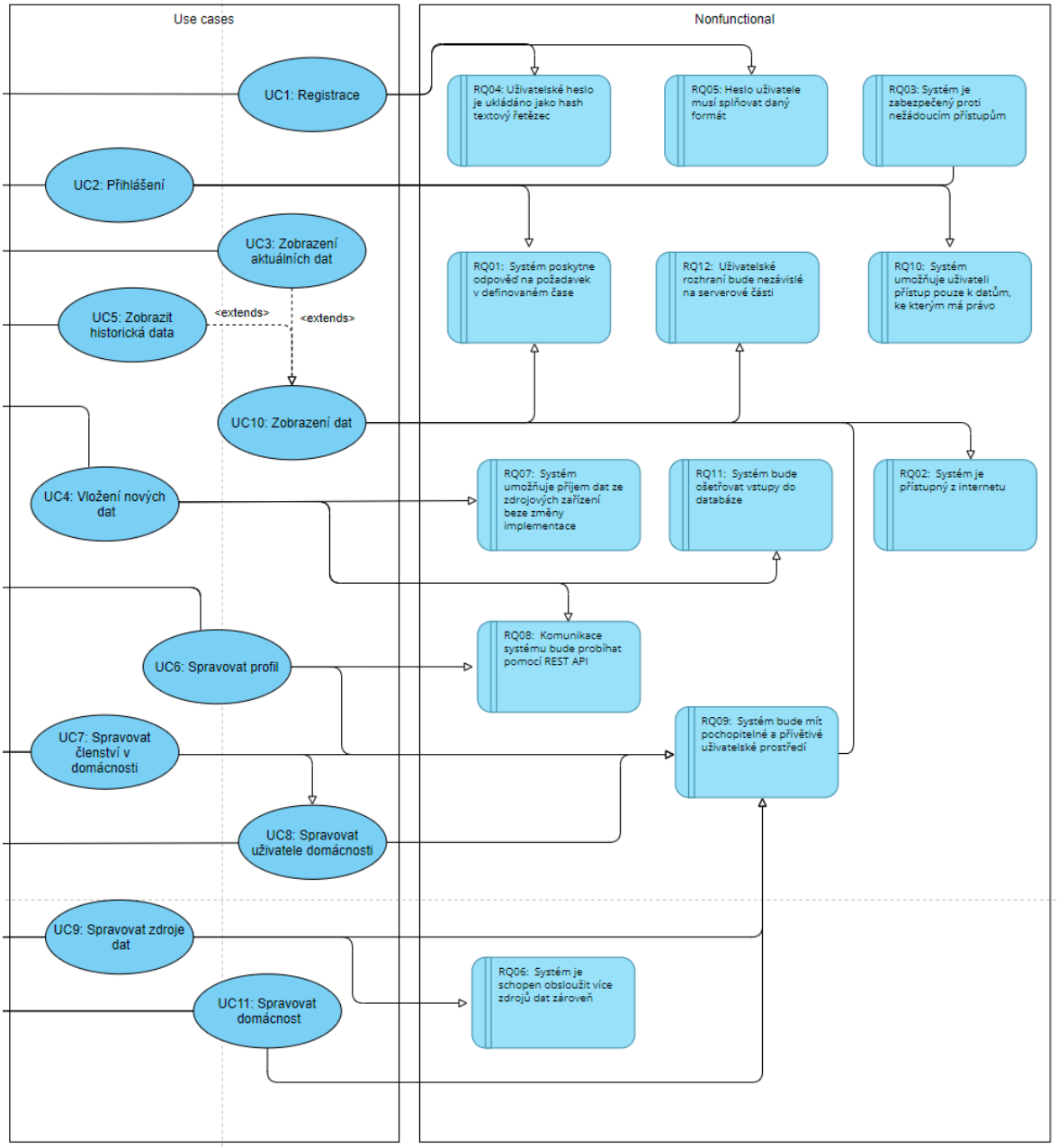
Každý požadavek musí být pokryt alespoň jedním případem užití. K vyjádření tohoto stavu lze využít dvou indikátorů. Jedním z nich je model pokrytí, který graficky zobrazuje vztahy mezi objekty v modelu. Zahrnuty jsou zde i vztahy mezi případy užití, které jsou zobrazeny v předchozí části.

Druhou možností je matice pokrytí, která zobrazuje pokrytí v tabulce, kdy na jedné ose jsou funkční a / nebo nefunkční požadavky a na druhé případy použití. Pokud případ užití pokrývá požadavek, je toto vyznačeno na průniku příslušného řádku a sloupce tabulky.

6.3.3.1 Model pokrytí požadavků



Obrázek 15 Model pokrytí požadavků - funkční požadavky



Obrázek 16 Model pokrytí požadavků - nefunkční požadavky

6.3.3.2 Matice pokrytí požadavků

	RQ1	RQ2	RQ3	RQ4	RQ5	RQ6	RQ7	RQ8	RQ9	RQ10	RQ11	RQ12	RQ13	RQ14	RQ15
UC1	■														
UC2		■													
UC3			■												
UC4															
UC5				■											
UC6										■		■			
UC7					■	■									
UC8							■								
UC9								■	■				■		
UC10															
UC11														■	■

Tabulka 1 Matice pokrytí požadavků - funkční požadavky

	RQ1	RQ2	RQ3	RQ4	RQ5	RQ6	RQ7	RQ8	RQ9	RQ10	RQ11	RQ12
UC1				■	■							
UC2	■		■							■		
UC3												
UC4							■	■			■	
UC5												
UC6								■	■			
UC7									■			
UC8									■			
UC9						■			■			
UC10	■	■										■
UC11									■			

Tabulka 2 Matice pokrytí požadavků - nefunkční požadavky

6.3.4 Modely tříd

Modely jednotlivých tříd je možné rozdělit na několik „kategorií“. Konkrétně na modely, které reprezentují obsah databáze a modely používané na backendu aplikace, sloužící k manipulaci s daty. Další kategorie modelů je určena na zobrazování dat na straně klienta, a nakonec ty, které se používají k přenosu dat, hlavně mezi serverovou částí a klientskou částí aplikace, nazývané jako„ (data) transfer objects“.

Během výměny dat mezi jednotlivými druhy modelů se můžou některé parametry buď přidat nebo vynechat, čímž dochází buď k „odfiltrování“ nechtěných a nepotřebných informací nebo k zahrnutí dat, která jsou v těchto modelech potřeba. Nechtěnými údaji se myslí

například citlivá data, která by neměla opustit backendovou část aplikace, například hesla nebo některé osobní údaje.

6.3.4.1 Databáze

Součástí většiny softwarových systémů je i databázový systém určený pro ukládání většího množství dat. Není tomu jinak ani u tohoto systému. Samotná databáze bude hostovaná na stejném serveru / zařízení, jako aplikace samotná, tedy na mini počítači Raspberry Pi.

Co se týká obsahu databáze, bude obsahovat několik různých tabulek vzájemně propojených pomocí definovaných vztahů. O vytváření těchto tabulek a vztahů se bude starat aplikace samotná pomocí Entity frameworku. Ten bude zajišťovat jak generování objektů do databáze, tak i následné plnění tabulek potřebnými daty.

Výstup databáze v aplikaci bude také brán jako model. K jejich naplňování budou využívány tzv. „repository" třídy, zajišťující komunikaci s backendem a databázovým systémem. Získaná data budou předávána dále do dalších modelů, až se eventuálně dostanou až na klientskou část aplikace k uživateli.

6.3.4.2 Backend modely

Tyto modely slouží pro ukládání a následnou manipulaci s daty v serverové části aplikace. Naplněny jsou daty, které se získají z databáze a po jejich zpracování jsou předávány dalším částem aplikace.

6.3.4.3 DTO modely

Jak už bylo zmíněno, tyto modely slouží k předávání dat ze serverové části na klientskou a naopak. Jednotlivé modely se dají dále dělit na „Request" požadavky, které „žádají" o nějaká data nebo akci, nebo „Response" požadavky neboli odpovědi na předchozí požadavky.

Žádosti samotné většinou obsahují velmi málo dat, která jsou ale důležitá ke správnému získání požadovaných informací. Konkrétně se může jednat o požadavek na přihlášení, který obsahuje pouze uživatelské jméno a heslo, nebo dotaz na specifická data, identifikovaná pomocí jejich globálně unikátního identifikátoru (GUID).

Odpovědi na požadavky mohou zahrnovat větší množství dat, jelikož se většinou vrací více položek, které samotné mohou být poměrně rozsáhlé. Například při požadavku na získání všech uživatelů v nějaké domácností, kdy je uživateli vráceno pole, které se velikostně může pohybovat od jednotek až to desítky a stovky u některých systémů.

Samozřejmě velikost může být u obou typů modelů různá. Požadavky mohou být i pro vytvoření nových dat, kdy jejich obsahem bude více údajů. Odpovědi na druhou stranu mohou být i jednoduché, příkladem mohou být potvrzení úspěšnosti přidání nebo úpravy nějakých dat či záznamů.

6.3.4.4 Frontend modely

Poslední použitý typ modelů má podobné zaměření, jako modely na backendu. Slouží pro manipulaci s daty, které byly získány z DTO modelů a slouží pro předání dat funkcím a třídám, které obsažená data zobrazí uživateli. Podobně jako ostatní modely mohou tyto modely být obohaceny nebo ochuzeny o některé parametry, které jsou či nejsou pro zobrazování potřebné.

6.4 Frontend a backend

6.4.1 Klientská část / frontend

Hlavním úkolem klientské části systému je poskytnutí uživatelného rozhraní, pomocí kterého bude uživatel moci ovládat systém a zobrazovat shromážděná data. Úvodní stránka bude dostupná bez přihlášení a zobrazovat bude nejnovější meteorologická data z nejbližší meteostanice. Po přihlášení bude moci uživatel zobrazovat i historická data a dále interagovat se systémem a jeho částmi.

Implementace bude vytvořena pomocí technologie Blazor a psána v programovacím jazyku C#. Konkrétně se jedná o variantu Blazor WebAssembly, která dokáže pracovat bez přímé závislosti na serverové části, čímž ulehčuje zatížení serveru samotného na úkor mírného snížení výkonu na straně uživatele. Provedení bude také následovat návrhové vzory uvedené v teoretické části práce, jako je Atomic desing nebo Dependency Injection.

6.4.2 Serverová část / backend

Stejně jako klientská část bude i ta serverová vytvořena pomocí jazyku C#, konkrétně za pomoci ASP.NET Core frameworku. Hlavním účelem této části je poskytovat API přístupové body, pomocí kterých bude moci komunikovat s ostatními částmi systému, tedy s frontendem a také meteostanicemi, případně v další implementaci i jinými systémy nebo aplikacemi.

K hostování aplikace bude využit Kestrel [70], což je multiplatformní webový server, který je v základu používaný u ASP.NET Core projektu. Existuje zde i možnost využití alternativních serverových systémů, to by ale zahrnovalo složitější nastavování, přičemž server Kestrel je více než dostatečný pro účely této služby.

6.5 Meteostanice

V tomto systému je meteostanice hlavním a momentálně taky jediným zdrojem dat. Jedná se o hardwarové zařízení, složené ze senzorů a dalších komponent, které sbírá informace o okolí a následně je předává systému samotnému. Jako hlavní požadavek zde byla samostatnost a co nejmenší potřeba údržby nebo dalších zásahů do meteostanice, hlavně kvůli možnosti umístění tohoto zařízení v těžko dostupných místech.

6.5.1 Návrh konstrukce

Jako pomyslné srdce meteostanice byl vybrán mikrokontroler Arduino Nano 33 IoT, který byl navržen právě na využití v rámci Internetu věcí. Má nízkou spotřebu, dokáže komunikovat pomocí Bluetooth i Wi-Fi a má dostatečný počet vstupů a výstupů k připojení potřebných senzorů, přičemž stále zůstává dostatečné množství volných pro možnou rozšiřitelnost.

K němu je připojena řada senzorů, které zaznamenávají okolní podmínky a předávají je mikrokontroleru ke zpracování a následnému odeslání. Konkrétní příklady jsou popsány dále.

Meteostanici jako takové bude dodávána energie pomocí solárního panelu, který nabíjí jak samotný mikrokontroler, tak i baterie s dostatečnou kapacitou pro případy, kdy je venku tma nebo zataženo a solární panel nedokáže dodávat dostatečný proud.

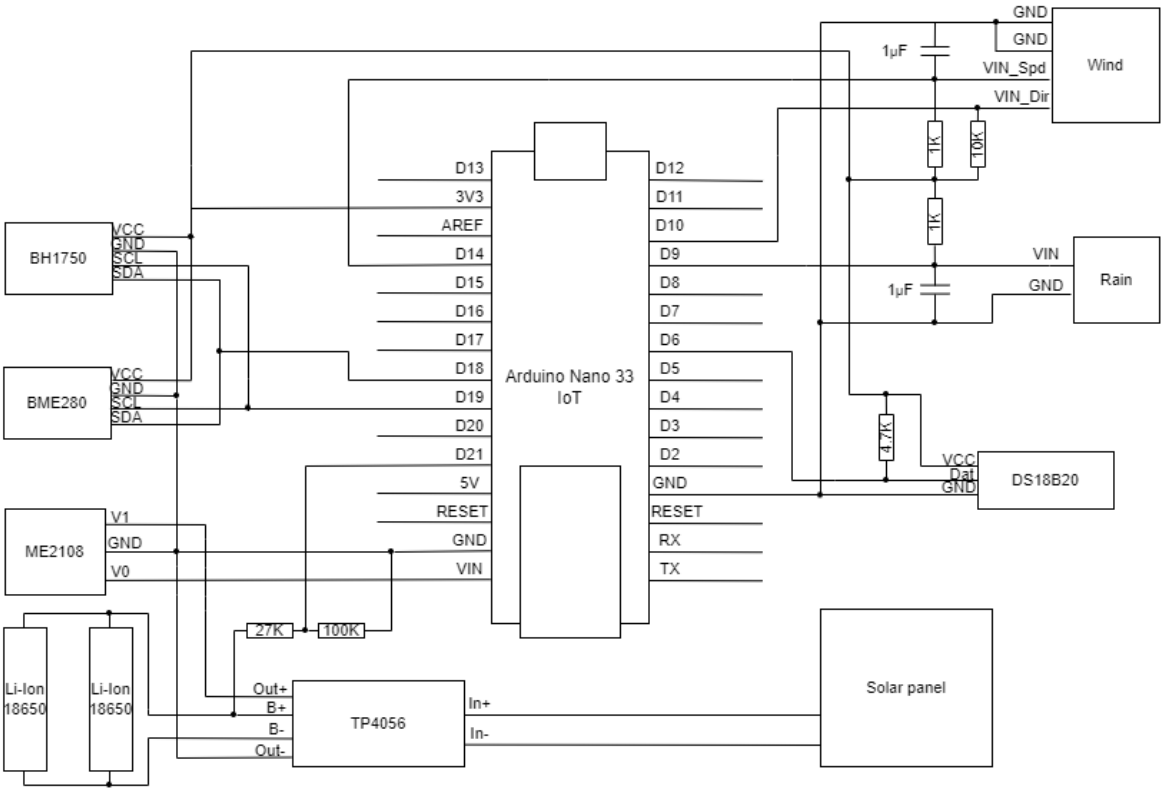
Jako dodatečnou součást měla být původně použita čtečka SD karet, na kterou by se zaznamenávaly data v případě výpadku spojení se serverem nebo jiným problémem. Kvůli problémům s implementací bylo ovšem od tohoto úmyslu odpuštěno a není zahrnut v rámci této práce.

Vnější konstrukce zahrnuje plastový radiační kryt pro senzor měření „externí“ teploty, krabičku na uložení kontroléru s bateriemi a některými senzory, a hliníkovou „kostru“, ke které budou pomocí plastových součástí, vytištěných na 3D tiskárně, přichyceny jednotlivé části.

6.5.2 Diagram

Na následujícím schématu je hrubé propojení jednotlivých komponent meteostanice mezi sebou. Hlavní komponentou je mikrokontroler, ke kterému jsou připojeny a se kterým komunikují či jinak spolupracují ostatní hardwarové části.

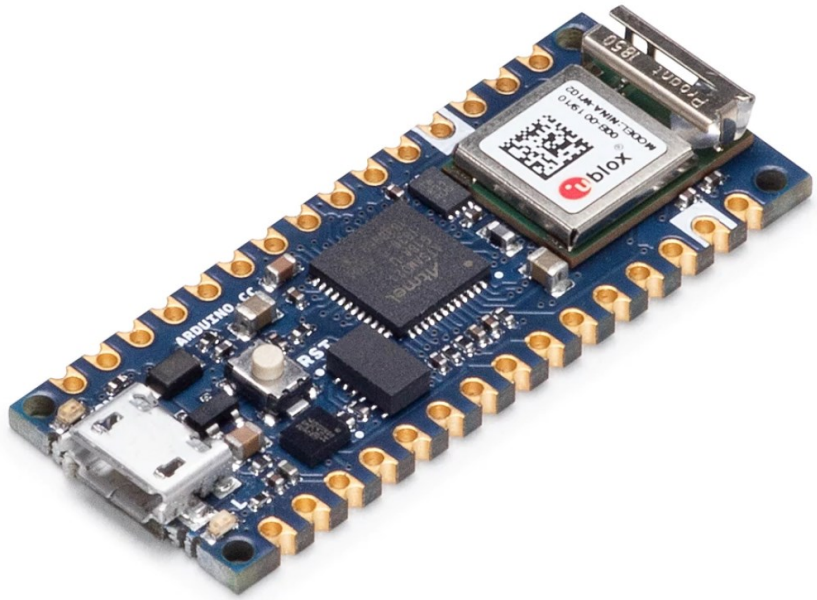
Jednotlivé komponenty lze rozdělit na dvě kategorie, dle jejich účelu. První jsou součásti napájení mikrokontroleru a ostatních částí meteostanice. Na obrázku je tato část ve spodní části. Druhým typem jsou pak samotné senzory, připojené přímo k mikrokontroleru, ze kterého také berou napájení o napětí 3.3 V.



Obrázek 17 Diagram zapojení meteostanice

6.5.3 Komponenty

Kromě již zmíněného mikrokontroleru ve formě Arduino Nano je součástí meteostanice celá řada dalších komponentů a senzorů. Co se provedení týče, jedná se o ověřené a funkční senzory, které jsou lehce dostupné, a i poměrně levné. Kromě dále uvedených příkladů je stále možné používat i alternativy s podobnou funckionalitou, které jsou například přesnější nebo výkonnější.



Obrázek 18 Arduino Nano 33 IoT [71]

6.5.3.1 DS18B20

Pro měření „externí“ teploty je použit tento číslicový teplotní senzor s více než dostatečným rozsahem a přesností. Ke komunikaci slouží sběrnice 1-Wire, která používá pouze jeden vodič pro přenos dat. Tato sběrnice byla navržena společností Dallas Semiconductor a slouží pro komunikaci zařízení nízkými rychlostmi. Protože se jedná o sběrnici, lze k jednomu vstupu zapojit více paralelních zařízení za sebou.

Označení „externí“ v našem případě znamená, že senzor je přímo vystaven povětrnostním podmínkám a chráněn před přímým slunečním zářením pomocí radiačního štítu.

6.5.3.2 BME280

Tento senzor oproti předchozímu dokáže zaznamenávat více rozdílných veličin. Slouží k měření „interní“ teploty, vlhkosti a také tlaku. Pro předávání informací používá sběrnici I^2C (známe i jako Two Wire Interface), která oproti dříve zmíněné sběrnici 1-Wire používá dva vodiče pro komunikaci a má vyšší rychlost na úkor menšího dosahu.

„Interní“ znamená umístění v krytém ochranném boxu, který pokud možno není vystaven přímému slunečnímu svitu a větru.

6.5.3.3 BH1750

Pro měření intenzity slunečního záření byl zvolen tento senzor, který má měřící rozpětí až do 65 535 lux, což odpovídá i integrovanému 16bitovému AD převodníku. Podobně jako předchozí senzor pro měření teploty, vlhkosti a tlaku používá sběrnici I^2C. Pro zajištění správného provozu má senzor nastavenou jinou adresu, čímž lze v rámci sběrnice komunikovat s více zařízeními.

6.5.3.4 Anemometr a korouhev

Sensory na měření intenzity a směru větru jsou původem náhradní díly pro komerční meteostanici, které se prodávají zvlášť. Oba pracují na principu spínání tzv. „jazýčkového kontaktu“. Spínač obsahuje dva tenké kovové jazýčky, které se po přiblížení zdroje magnetického pole spojí a vytvoří kontakt. U anemometru je to jeden spínač, který se sepne pokaždé, když dojde k otočení „rotorové“ části. Výsledná rychlost se vypočítá z počtu otáček za nějaký časový úsek. V případě korouhve se jedná o několik spínačů, každý připojený ke specifickému odporu, přičemž výsledný směr lze vypočítat podle výsledného napětí na výstupu, který je změněn oprotiv vstupnímu o hodnotu aktivních odporů.

Kromě varianty ve formě náhradních dílu lze využít i profesionální a více přesné senzory, ty jsou ovšem podstatně dražší a mohou fungovat na jiném principu, než je zde popsán. Samozřejmě je zde i varianta vytvoření vlastní verze, například za využití 3D tisku.

6.5.3.5 Srážkoměr

Stejně jako senzory na monitorování větru, i srážkoměr je prezentován jako náhradní díl. Pracuje také na podobném principu zahrnující spínací kontakt pro měření údajů, pomocí měření počtu překlopení misek uvnitř senzoru. Dešťová voda je sbírána do obdélníkové nádoby, která má odtok ve svažujícím se středu. Tato voda naplní misky a ty se při určitém naplnění překlopí, sepnou spínač a ten vyšle signál. Celková velikost srážek je následně vypočítána podle plochy nádoby a obsahu vody nutném pro překlopení misek.

Podobně jako u anemometru a korouhve existuje varianta tvorby vlastního senzoru, případně koupě dražšího a kvalitnějšího senzoru. Zde existují například varianty, které měří kapky na průhledné kopuli pomocí odrazu laserového paprsku.

6.5.3.6 Komponenty napájení

Arduino a další mikrokontrolery mají specifické požadavky na napájení. Meteostanice bude napájena pomocí solárního panelu, který má proměnlivou velikost výstupního napětí, a pomocí baterií, které se pro různé úrovně naplnění kapacity chovají podobně. Je tedy požadováno, aby byl proud vstupující do mikrokontroleru na určité úrovni, a pokud možno stabilní.

Použité Li-Ion články jsou náchylné na špatné zacházení, většinou týkající se nabíjení článků. Proto byl využita nabíječka TP4056, která přijímá vstup ze solárního panelu a má výstupy do baterií, které nabíjí a také samotný výstup proudu. Obvod zahrnuje i ochranu proti přílišnému vybití, nabití a také zkratu.

Zvolené Arduino má rozsah pro vstupní napětí mezi 4.5 a 21 V. Protože baterie mají při plné kapacitě výstupní napětí kolem 4.2 V, je samostatně nepoužitelné. Proto je využit tzv. měnič napětí, konkrétně ME2108, který dokáže zvýšit úroveň vstupního napětí na 5 V, které lze použít na napájení mikrokontroleru přes Vin port.

Pro jiné mikrokontrolery nemusí být tento obvod vhodný, například mají jiný rozsah vstupního napětí. Naštěstí existuje i řada alternativních obvodů s jinými parametry, proto není nutné se tohoto sestavení držet.

6.5.3.7 Další komponenty

Mezi další komponenty se řadí hlavně různé odpory a vodiče, které spojují jednotlivé části. Taktéž se sem dají zařadit například terminály pro připojení senzorů a vstupu ze solárního panelu, četně koncovek typu RJ15 pro připojení větrných senzorů a srážkoměru.

Je zde zahrnuta i PCB, tedy desku plošných spojů, která poskytuje jakýsi základ pro připojení jednotlivých komponent do jednoho celku. Tu lze buď vyrobit připájením komponent na jednoduchou děrovanou desku a pospojování datových a napájecích částí, nebo si podle diagramu nechat tuto desku na míru vyrobit u společnosti na to zaměřené. Vždy jde také ponechat obvod zapojený v tzv. breadboard, což je deska sloužící na prototypování různých obvodů a zařízení. Tato možnost ovšem není příliš doporučována, hlavně kvůli náchylnosti na možné rozpojení některé části a tím znefunkčnění celku.

6.6 Komunikace

Jak už bylo naznačeno, jako způsob komunikace byl zvolen protokol HTTP, tedy přenos informací přes internet. K tomu by se meteostanice připojovala pomocí Wi-Fi a na příslušný

API koncový bod na serveru odeslala naměřená data. Jelikož tento způsob komunikace je více náročný na energii, je žádoucí, aby byla komunikace aktivní jen v okamžiku odesílání dat. Po dobu nečinnosti, potažmo čekání na další interval měření, by byl Wi-Fi modul odpojený od sítě a mikrokontroler v režimu nízké spotřeby.

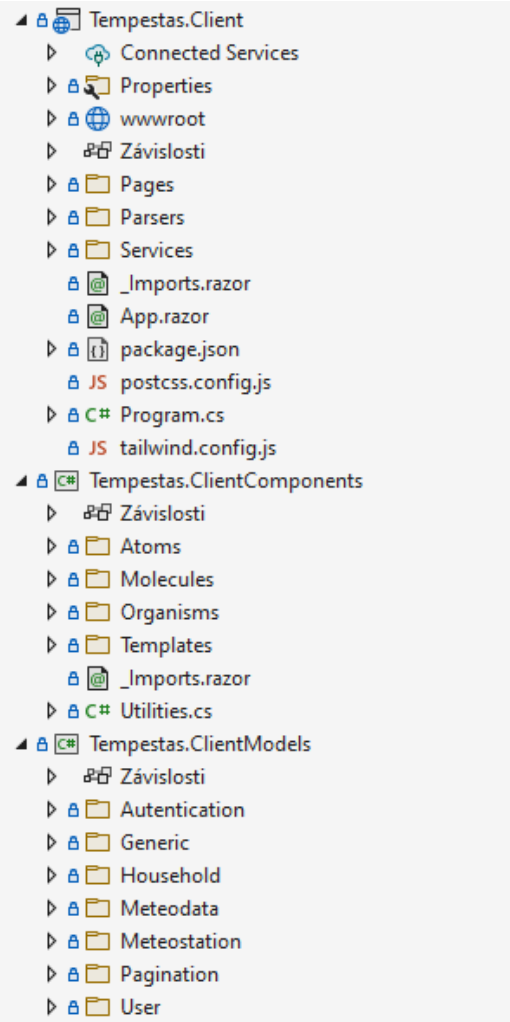
Jako alternativu lze použít drátové připojení k internetu. Při komunikaci tímto způsobem by odpadala potřeba šetřit energii, protože by bylo možné implementovat napájení pomocí PoE, tedy přenášením napětí přímo přes ethernetový kabel. Tím by odpadla i nutnost solárního panelu a také baterií, případně by to umožnilo provést implementaci tak, že by zařízení bylo dostupné i z vnějšku, což by mohlo sloužit pro získání aktuálnějších dat mimo stanovený interval.

7 IMPLEMENTACE A ŘEŠENÍ

V této části je popsán vývoj systému a některé překážky, které během tohoto procesu nastaly. To se týká jak softwarových částí, které byly většinou vyřešeny bez větších problémů, a hardwarové části, která představovala v některých ohledech o něco větší výzvu.

7.1 Frontend

Klientská část aplikace je rozdělena na tři projekty, každý starající se o jinou činnost a jinou oblast. Konkrétně jsou zde projekty Client, který je pomyslným základním projektem, dále projekt Components, obsahující komponenty Atomického designu, a nakonec projekt Models zahrnující modely používané v rámci klientské části aplikace. Toto rozdělení do jisté míry odpovídá návrhové architektuře či vzoru Model-View-Controller.



Obrázek 19 Struktura řešení - Frontend

7.1.1 Klient

V prvním projektu v rámci frontendové části aplikace se nachází hlavní logika, starající se o získávání dat a jejich převedení do podoby, kterou je možné zobrazit uživateli. Také je zde nacházejí konfigurační soubory, statické zdroje stránek a vstupní bod programu.

7.1.1.1 Kořenový adresář

Ve složce wwwroot jsou ukládány statické soubory používané na webových stránkách. Může se jednat o kaskádové styly, různé obrázky, skripty a jiné zdroje, četně přímo některých webových stránek. K těmto zdrojům se dá většinou dostat přímo z prohlížeče tím, že zadáme cestu, na které je daný zdroj fyzicky umístěn v projektu, začínaje samotným kořenovým adresářem.

Také se v této složce nachází index.html, neboli soubor, v rámci kterého dochází k načtení Blazor aplikace samotné, četně všech potřebných zdrojů a referencí. Místo toho, aby byly jednotlivé stránky ve svých .html souborech, dochází ke změnám obsahu právě indexového souboru. Tento přístup má řadu výhod, nejvíce pro uživatele viditelnou je plynulý přechod mezi stránkami.

Dalšími důležitými sobory jsou zde hlavní kaskádové styly, které jsou výstupem frameworku Tailwind a mimo jiné i šablona pro vytváření kódu pro meteostanice. Také lze zmínit, že se zde také nachází soubor, který byl použit pro ověření vlastnictví použitého serveru pro získání SSL certifikátu, umožňujícího vytvoření zabezpečené komunikace mezi klientem a serverem.

7.1.1.2 Stránky

Tato část slouží jako pomyslné View(s), tedy pohledy, v rámci návrhového vzoru MVC. Jejich cílem je získat data pomocí volání služeb a následně je zobrazit ve vhodné formě uživateli. Pro zobrazování dat používají komponenty vytvořené pomocí vzoru Atomic design, proto jsou z tohoto pohledu soubory poměrně prázdné. Hlavní částí je zde získávání a zobrazování dat získaných ze služeb aplikace. Při načtení stránky nebo při provedení nějaké akce se zde parametry předají službám, které onu akci provedou a výsledek odešlou zpět stránce, která výsledek předá jednotlivým komponentám, které ho zobrazí.

Každá tato stránka má i svou cestu, na které si ji může uživatel zobrazit. Ty lze definovat buď přímo nebo, jako v našem případě, pomocí atributu, který odkazuje na statické

konstanty ve sdíleném projektu. To je z důvodu zajištění konzistence a ulehčení případného upravování, při kterém by bylo nutné měnit každou tuto cestu zvlášť.

```
@attribute [Microsoft.AspNetCore.Components.RouteAttribute(EndpointConstants.UserRegister)]

@using Tempestas.Client.Services.Data.Autentications
@using Tempestas.ClientModels.Autentication

<PageTitle>@Miscellaneous.PageName - Register</PageTitle>

<RegisterPage RegisterUser="(registration) => RegisterUser(registration)"
              RegisterResponseErrors="@RegisterResponse.ParseErrors()" ></RegisterPage>

@code {
    [Inject]
    IAuthenticationService AuthenticationService { get; set; }

    [Inject]
    public NavigationManager NavigationManager { get; set; }


    private RegisterResponse RegisterResponse = new RegisterResponse();

    public async Task RegisterUser(RegisterRequest registration)
    {
        var result = await AuthenticationService.Register(registration);

        if (!result.IsSuccessfulRegistration)
        {
            RegisterResponse = result;

            StateHasChanged();
        }
        else
        {
            NavigationManager.NavigateTo(EndpointConstants.UserLogin);
        }
    }
}
```

Obrázek 20 Aplikace - Stránka registrace

7.1.1.3 Služby

Tato část slouží hlavně ke získávání dat, potažmo obsahuje volání koncových bodů v serverové části aplikace. Data přejatá z odpovědí na tyto volání se zpracují a převedou do frontendových modelů, které jsou následně použity jako zdroj pro jednotlivé komponenty. Obdobná služba existuje pro každý typ dat, se kterými systém pracuje.

Většina služeb je v projektu používána jako Dependency Injection takovým způsobem, že jednotlivé služby se mohu využívat navzájem bez nutnosti složitých vytváření vazeb a instancí. K tomu je potřeba, aby každá služba měla deklarovaný Interface, který slouží jako jakási předloha, určující, co má daná služba obsahovat za třídy a jak má obecně vypadat. V souboru, ve kterém probíhá vytváření instancí těchto služeb, je následně definováno, jaká

služba je spojena, s jakým rozhraním. Následně už stačí ve třídě, ve které chci danou službu používat, definovat ve vstupních parametrech konstruktoru, že chci používat tuto službu, a zbytek je zajištěn automaticky, bez dalšího zásahu uživatele.

```
namespace Tempestas.ServerCore.Services.CommandServices.AuthenticationService
{
    Počet odkazů: 2
    public class AuthenticationCommandService : IAuthenticationCommandService
    {
        private readonly IApplicationUserManager _applicationUserManager;
        private readonly IUserRepository _userRepository;
        private readonly IJwtService _jwtService;

        Počet odkazů: 0
        public AuthenticationCommandService(IApplicationUserManager applicationUserManager,
                                            IUserRepository userRepository,
                                            IJwtService jwtService)
        {
            _applicationUserManager = applicationUserManager;
            _userRepository = userRepository;

            _jwtService = jwtService;
        }

        Počet odkazů: 3
        public async Task<RegisterResponse> Register(UserRegisterRequest userRegisterRequest, string role)
        {
            var user = new User { UserName = userRegisterRequest.Username };

            var result = await _applicationUserManager.CreateAsync(user, userRegisterRequest.Password);
            if (!result.Succeeded)
            {
                var errors = result.Errors.Select(e => e.Description);

                return new RegisterResponse { Errors = result.Errors };
            }

            await _applicationUserManager.AddToRoleAsync(user, role);

            var getUser = await _applicationUserManager.FindByNameAsync(user.UserName);

            return new RegisterResponse { CreatedUserId = getUser.Id, IsSuccessfulRegistration = true };
        }

        Počet odkazů: 2
        public async Task<LoginResponse> Login(UserLoginRequest userLoginRequest)...

        Počet odkazů: 2
        public async Task<ChangePasswordResponse> ChangePassword(UserChangePasswordRequest userChangePasswordRequest, Guid requestingUserId)...
    }
}
```

Obrázek 21 Aplikace - Autentizační služba na straně klienta

V této sekci jsou definovány také jiné služby. Jako příklad lze uvést služba, která slouží na vytváření konfiguračních souborů pro meteostanici jako takovou. Data zadaná uživatelem jsou použita pro nahrazení konkrétních parametrů v šabloně, která se poté nahrává do mikrokontroleru a slouží k ovládání meteostanice a umožňuje její komunikaci s naší aplikací. Další službou je třída starající se o zpracovávání JWT tokenů. Jejich podstatou je získání tohoto tokenu z odpovědi při přihlášení, jeho uložení do Session Storage webového prohlížeče a následné použití pro autorizaci další komunikace. Oproti Local Storage jsou zde data uložena pouze po dobu, kdy je stránka aktivní. Jakmile tedy dojde k zavření stránky, toto úložiště je smazáno a s ním i token, čímž “zaniká” i současné přihlášení. Další variantou může být ukládání i do tzv. Cookies, což může mít své výhody i nevýhody.

7.1.1.4 Program.cs

Tato třída slouží jako vstupní bod do programu. Aplikace tady začíná a také tady za normálních okolností i končí. Kromě sestavování aplikace jsou zde definovány i funkce poskytnuté NuGet balíčky a také přiřazení služeb a tříd, sloužící pro implementaci vzoru vkládání závislostí. Po určení všech parametrů je zde aplikace sestavena a nakonec i spuštěna.

7.1.2 Modely

Modely slouží jako reprezentace dat, s kterými aplikace pracuje. Data, která jsou přijímána k serverové části aplikace jsou uložena do těchto modelů a modely samotné potom předány až k samotným stránkám a data v nich obsažená prezentována uživateli.

Význam a použití jednotlivých modelů již byl dříve v této práci popsán. Co se jednotlivých druhů týče, kromě modelů pro posílání a přijímání, potažmo zobrazování, jsou zde i generické modely, používané jako potvrzení úspěšnosti nějaké akce, ke které jinak nepřijímáme žádná data. Také jsou zde modely, používané v rámci stránkování, definující parametry, jako je aktuální stránka, celkový počet a rozsah stránek.

7.1.3 Komponenty

Pomocí komponentů je tvořen samotný vzhled stránek a jednotlivých elementů na nich. Jak už bylo zmíněno, tato část využívá návrhovou metodologii, zaměřenou na rozdělení komponent stránky na jednotlivé prvky, pomocí kterých se bude výsledný vzhled stránky vytvářet a skládat. Je asi nutné zmínit, že Atomic design jako takový je poměrně subjektivní, co se týče implementace a řazení jednotlivých prvků. Příkladem může být tlačítko. V případě, že bude tato komponenta obsahovat rozsáhlejší logiku, vznikne dilema, jestli se jedná o Atom nebo už o Molekulu.

```
@using Microsoft.AspNetCore.Identity
@using Tempestas.ClientModels.Autentication

<LoginRegister >
    <p class="uppercase text-3xl py-12">Register</p>

    <TextInput Value="@_registerObject.Username"
               OnChange="value => {_registerObject.Username = value; StateHasChanged();}"
               ErrorValue="@RegisterResponseErrors.Username.FirstOrDefault()"
               Type="text"
               Placeholder="Username" />
    <TextInput Value="@_registerObject.Password"
               OnChange="value => {_registerObject.Password = value; StateHasChanged();}"
               ErrorValue="@RegisterResponseErrors.Password.FirstOrDefault()"
               Type="password"
               Placeholder="Password" />
    <TextInput Value="@_registerObject.ConfirmPassword"
               OnChange="value => {_registerObject.ConfirmPassword = value; StateHasChanged();}"
               ErrorValue="@RegisterResponseErrors.ConfirmPassword.FirstOrDefault()"
               Type="password"
               Placeholder="Password again"
               OnKeyPress="() => TryRegisterUser()" />

    <Button OnClickAsync="_ => TryRegisterUser()" Text="Register" IsDisablable="true" Color="primary-bg-btn w-full h-12" />

</LoginRegister>
```

Obrázek 22 Aplikace - Šablona registrace - část 1

Jednotlivé komponenty mají v sobě definovanou i základní logiku, která upravuje jejich vzhled a funkce podle toho, v jakém kontextu je komponent použit. Jako příklad zde můžeme uvést tlačítko. To má definovaný výchozí vzhled a základní funkcionalitu. Pro konkrétní použití je ovšem nutné poskytnout této komponentě další informace, v případě tlačítka se jedná hlavně o adresu, na kterou má uživatele přesměrovat, nebo o akci, kterou má po stisku provést. Taktéž může být tlačítko používáno v rámci formuláře pro odeslání požadavku, u kterého je vhodné počkat na výsledek této akce ze strany serveru. Můžeme proto tlačítko třeba „vypnout“ a tím zabránit jeho další aktivaci. Tlačítko také může být také používáno s různými styly, případně může obsahovat nějaké další vizuální prvky, symbolizující například čekání na vyřízení požadavku nebo položku v rámci menu aplikace. Možností, jak tlačítko tedy upravit pro naše specifické požadavky je celá řada, přičemž výhodou je hlavně jeho univerzálnost, neboť ke většině aplikací lze do určité míry použít stejnou komponentu bez úprav v implementaci.

```
@code {
    private RegisterRequest _registerObject = new RegisterRequest();

    [Parameter]
    public RegisterResponseErrors RegisterResponseErrors { get; set; } = new RegisterResponseErrors();

    [Parameter]
    public Func<RegisterRequest, Task> RegisterUser { get; set; }

    public async Task<bool> TryRegisterUser()
    {
        var errors = new RegisterResponseErrors();

        if (string.IsNullOrEmpty(_registerObject.Username))
        {
            errors.Username.Add("Username is required");
        }

        if (string.IsNullOrEmpty(_registerObject.Password))
        {...
        }

        if (string.IsNullOrEmpty(_registerObject.ConfirmPassword))
        {...
        }

        if (_registerObject.Password != _registerObject.ConfirmPassword)
        {...
        }

        if (errors.Username.Count is 0 && errors.Password.Count is 0 && errors.ConfirmPassword.Count is 0)
        {...
        }
        else
        {...
        }

        StateHasChanged();

        return false;
    }
}
```

Obrázek 23 Aplikace - Šablona registrace - část 2

7.1.3.1 Atomy

Prvním a základním prvkem jsou atomy neboli pomyslné základní stavební kostky stránky. Konkrétně jsou zde definovány prvky, se kterými uživatel interaguje, jako například tlačítka, vstupní pole, textové a jiné vstupy. Také jsou v této části definovány některé obecnější prvky, kupříkladu komponenta "stránky", která určuje pozici všech dalších prvků v rámci stránky.

Zahrnuty jsou zde i některé informační prvky. Je zde definován „spinner“, což je rotující neúplné kolečko, symbolizující načítání nebo čekání na nějakou akci. Také je zde i tzv. „toast“, což je prvek, který je využíván k poskytnutí zpětné vazby uživateli ohledně nějaké akce, která byla provedena. V levém horním rohu se na krátkou chvíli zobrazí informace o úspěchu, případně i o selhání nějaké akce.

V neposlední řadě obsahuje složka s atomy podsložku, která má v sobě komponenty, definující obsah tabulek v aplikaci. Ty přijímají jako parametr pole prvků daného modelu, který převedou do podoby těla tabulek. Teoreticky zde existuje možnost vytvoření jedné univerzální komponenty pro všechny tabulky, ovšem tvorba takové komponenty by zabrala mnoho času, který by neospravedlňoval případnou přidanou hodnotu.

7.1.3.2 Molekuly

Molekuly oproti atomům tvoří komplexnější struktury, složené převážně z atomů samotných. Převážně jsou zaměřeny na tabulky, konkrétně na přidání záhlaví k atomu s tělem a rozšíření těchto tabulek o tzv. stránkování. Tím je umožněno načíst data po částech místo všech najednou, čímž by utrpěl výkon a rychlost načítání stránky. Kromě komponentů tabulek jsou zde například seskupení několika tlačítek do společné skupiny, definování záhlaví stránky a bočního menu stránky.

Oproti předchozím komponentám, které se zabývali v podstatě výhradně vzhledem nebo interakcí s uživatelem, se zde nachází i komponenta, jejíž cílem je kontrola autorizace uživatele v rámci stránky, na které je komponenta použita. Při přihlášení přijímá uživatel autorizační token, který obsahuje řadu užitečných parametrů, zahrnujíc i role daného uživatele. Pokud je tato role definována, zmíněná komponenta umožní uživateli přístup a obsah stránky se zobrazí. Jestliže uživatel tento token nemá, nebo není validní, je mu zobrazeno upozornění o neautorizovaném přístupu. Díky charakteristice frameworku použitého na tvorbu klientské části aplikace lze toto zabezpečení obejít, ovšem ve výsledku to nebude mít vliv na zabezpečení aplikace jako takové, jelikož kromě ověřování u klienta probíhá kontrola i na serverové části, kterou již nelze tak lehce obejít.

7.1.3.3 Organismy

Další úrovní vzoru Atomic design jsou organismy, které opět kombinují nižší prvky do vyšších a komplexnějších. Jsou zde komponenty, sloužící pro vytvoření základu formulářů určených k přihlašování a registraci, které mají kvůli své podobnosti společný základ. Dále je zde definovaný modál, což je v podstatě forma vyskakovacího okna či dialogu, sloužící pro vkládání dat, a nakonec komponenta zobrazující současná data v rámci úvodní stránky a následně i Dashboardu.

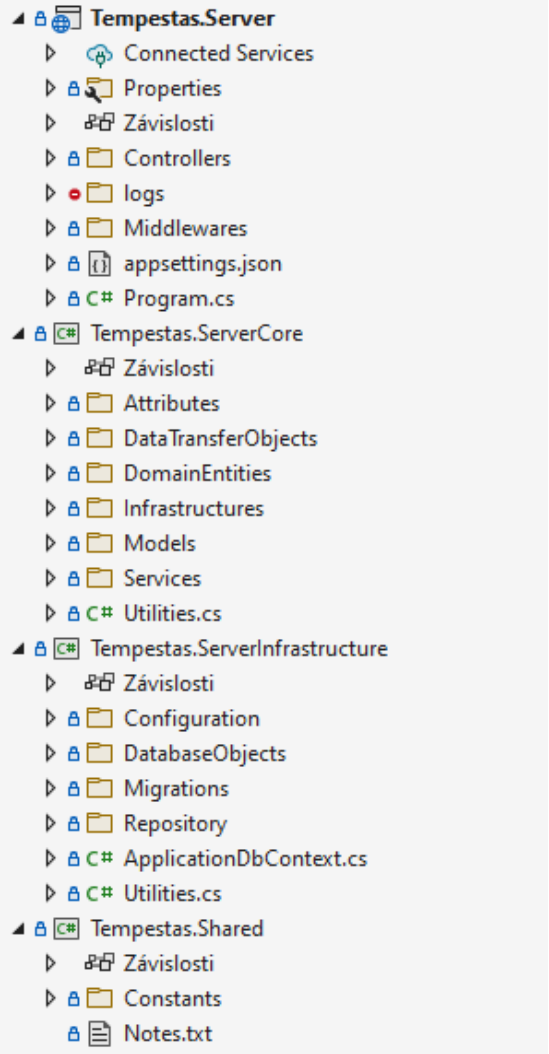
7.1.3.4 Šablony

Poslední částí jsou šablony, které pomocí přechozích částí vytváří finální podobu stránek. Jsou rozděleny na jednotlivé kategorie, například na autentizaci, zahrnující stránky pro registraci a přihlášení, správu meteostanic a domácností, nebo meteorologická data, kde jsou zahrnuty jak současná, tak historická data. Výsledné komponenty jsou poté vyžívány stránkami v Client části aplikace. Do těchto šablon jsou předána data, která jsou posléze upravena a následně zobrazena uživateli. Komponenty obecně také umožňují volání funkcí v nadřazených komponentách, čímž jsou předávány interakce se stránkou jako takovou, případně následné odpovědi na tyto akce.

7.2 Backend

Backend, podobně jako frontend, je rozdělen na několik částí v souladu s návrhovým vzorem Clean Architecture. API, které zajišťuje přístup k aplikaci zvenčí, Core, zahrnující služby, starající se o logiku, a modely, a nakonec Infrastucture, sloužící pro přístup k databázi a k jejímu obsahu.

Díky implementaci v rámci C# a projektů jako takových je zde do určité míry přímo Clean architektura vynucována. Projekty totiž umožňují na sebe odkazovat pouze jedním směrem, nemůže tedy například projekt Server odkazovat na projekt Core a zároveň projekt Core odkazovat na Server. Tím by vznikala cyklická závislost, která není u C# povolena.



Obrázek 24 Struktura řešení - Backend

7.2.1 API

Tento projekt zahrnuje samotné přístupové body a kontrolery, pomocí kterých dochází ke komunikaci vnějších služeb, jako je klientská část a meteostanice, se serverem. Jedná se také to pomyslný „hlavní“ projekt v rámci celého systému, protože slouží k jejímu spouštění po tom, co je projekt sestaven nebo nasazen.

7.2.1.1 Koncové body / kontrolery

Kontrolery jsou třídy, které v sobě mají definovány funkce, které slouží jako koncové body pro přístup z vnějšku aplikace. Rozdělené jsou podobně, jako ostatní části systému, podle jejich zaměření a účelu. Konkrétně jsou to například kontrolery pro autentizaci, správu uživatelů, domácností, meteostanic a jejich dat.

```
namespace Tempestas.Server.Controllers
{
    /// <summary>
    /// Controller used for registration, login and change password operations
    /// </summary>
    [ApiController]
    Počet odkazů: 1
    public class AuthenticationController : Controller
    {
        private readonly IAuthenticationCommandService _authenticationCommandService;
        private readonly IJwtService _jwtService;

        Počet odkazů: 0
        public AuthenticationController(IAuthenticationCommandService authenticationCommandService, IJwtService jwtService)
        {
            _authenticationCommandService = authenticationCommandService;
            _jwtService = jwtService;
        }

        [HttpPost(ApiEndpointConstants.RegisterUser)]
        Počet odkazů: 0
        public async Task<IActionResult> Register([FromBody] UserRegisterRequest userForRegistration)
        {
            var result = await _authenticationCommandService.Register(userForRegistration, RolesConstants.User);

            return result.IsSuccessfulRegistration? Ok(result) : BadRequest(result);
        }

        [HttpPost(ApiEndpointConstants.LoginUser)]
        Počet odkazů: 0
        public async Task<IActionResult> Login([FromBody] UserLoginRequest userLogin)...

        [HttpPost(ApiEndpointConstants.ChangePassword)]
        [Authorize]
        Počet odkazů: 0
        public async Task<IActionResult> ChangePassword([FromBody] UserChangePasswordRequest userChangePassword)...
    }
}
```

Obrázek 25 Aplikace - Kontroler autentizace

Každá funkce je definována s atributem, který určuje cestu, ze které je daný koncový bod dostupný. Ty jsou podobně, jako u stránek na frontendu, definovány ve sdíleném projektu jako konstanty. V tomto atributu je také určeno, o jaký typ koncového bodu se jedná, potažmo jestli jde o funkci ke získávání dat, k čemuž slouží GET, k přidávání dat pomocí POST a PUT, a také k mazání dat za použití DELETE.

Funkce mají také definovány vstupní parametry a návratovou hodnotu. Vstupy jsou data, která buď specifikují parametry, podle kterých chceme získat požadovaná data, nebo informace sloužící k vytvoření či změnění dat v systému. Návratovými hodnotami jsou buď generické odpovědi, které informují o úspěchu či neúspěchu akce, nebo objekty pro přenos vyžádaných dat.

Většina funkcí v těchto třídách pouze předává přijaté informace dál ke službám, některé ovšem vykonávají i další operace. Ku příkladu u funkce pro vytvoření domácnosti dochází kromě vytvoření této domácnosti i k přidání uživatele, který ji vytvořil, do této domácnosti. Následně dochází ke změně jeho role v této domácnosti na správce, a nakonec i k potvrzení pozvánky do domácnosti, kterou by jinak musel potvrdit manuálně v příslušné sekci.

7.2.1.2 Middlewares

Middleware jsou v ASP.NET Core součástí tzv. pipeline. Jedná se o řadu speciálních delegátů, které nějak upravují či kontrolují příchozí a odchozí data aplikace. Příchozí požadavky vstupují do této pipeline a postupně prochází všemi jednotlivými middleware. Samotná posloupnost je stanovena ve hlavním souboru aplikace, Program.cs. [72]

V základu obsahuje framework řadu těchto služeb, každá starající se o jinou funkci. Definuje se s jejich pomocí například použití statických souborů, přidání směrování v rámci aplikace, inicializace Swaggeru a taktéž autentizace a autorizace.

Zabezpečení je částečně prováděno vlastní službou, která se stará o ověřování JWT tokenů, pomocí kterých autorizuje uživatel své akce. Služba získá token z hlavičky příchozího dotazu a pomocí služby, zaměřené i na další operace s těmito tokeny, ověří, zda byl token vydán touto aplikací a zda je platný. Pokud platný je, předá služba dalším funkcím Id uživatele, kterému token náleží.

7.2.1.3 Program.cs a appsettings.json

Obsahem se tato třída příliš neliší od její verze v klientské části. Kromě konfigurace Dependency Injection, sestavování a spouštění programu má tato varianta navíc definované i jednotlivé middleware a jejich pořadí. Navíc jsou zde i funkce pro zaznamenávání provozu aplikace, nebo i pro získání přihlašovacích údajů do databáze a dalších služeb ze souborů s konfigurací.

Pro konfiguraci aplikace je využíván soubor appsettings. Ten je ve formátu .json a má definovány parametry, kterých využívá program během svého běhu. Nejdůležitějšími parametry je zde nastavení JWT tokenů s parametry jako šifrovací klíč, a také konfigurace serveru, používaná při nasazení systému. To primárně zahrnuje cestu k certifikátům, používaných pro vytvoření bezpečného spojení pomocí HTTPS. Posledním důležitým parametrem je textový řetězec, používaný k připojení do databáze, obsahující adresu serveru s databází a přihlašovací údaje k účtu v rámci databázového systému. Většina těchto parametrů kvůli bezpečnosti chybí v základní implementaci a je nutné je během nasazení aplikace doplnit.

7.2.2 Core

Core projekt obsahuje hlavní logiku aplikace ve formě služeb, které jsou volány z API projektu. Součástí projektu jsou taktéž modely a entity, definující funkcionalitu a použití aplikace. Taktéž jsou zde i rozhraní určené pro Infrastructure projekt, neboť je nutné, aby k nim

měly přístup i funkce a třídy z jiných vrstev projektu. Tato implementace také nenarušuje vzor Clean architecture, oproti případu, kdyby byla tato rozhraní v projektu Infrastructure.

7.2.2.1 Služby

Asi nejdůležitější součástí tohoto projektu jsou služby, složené z tříd a funkcí, které se starají o zpracovávání požadavků z API projektu a získávání dat z projektu Infrastructure. Kromě „obecných“ služeb jsou ostatní rozděleny dvou složek, Command a Query, obsahující služby rozdělené podle toho, jestli se zabývají získáváním dat nebo vytvářením a prací s nimi. Všechny třídy mají také deklarovaný své rozhraní, hlavně kvůli implementaci v rámci DI.

Služby nezařazené do těchto složek se konkrétně starají o poskytování času pro všechny ostatní služby a o práci s JWT tokeny, což zahrnuje primárně jejich vytváření a také následné ověřování z hlediska pravosti.

```
namespace Tempestas.ServerCore.Services.CommandServices.AuthenticationService
{
    Počet odkazů: 2
    public class AuthenticationCommandService : IAuthenticationCommandService
    {
        private readonly IApplicationUserManager _applicationUserManager;
        private readonly IUserRepository _userRepository;
        private readonly IJwtService _jwtService;

        Počet odkazů: 0
        public AuthenticationCommandService(IApplicationUserManager applicationUserManager,
                                            IUserRepository userRepository,
                                            IJwtService jwtService)
        {
            _applicationUserManager = applicationUserManager;
            _userRepository = userRepository;

            _jwtService = jwtService;
        }

        Počet odkazů: 3
        public async Task<RegisterResponse> Register(UserRegisterRequest userRegisterRequest, string role)
        {
            var user = new User { UserName = userRegisterRequest.Username };

            var result = await _applicationUserManager.CreateAsync(user, userRegisterRequest.Password);
            if (!result.Succeeded)
            {
                var errors = result.Errors.Select(e => e.Description);

                return new RegisterResponse { Errors = result.Errors };
            }

            await _applicationUserManager.AddToRoleAsync(user, role);

            var getUser = await _applicationUserManager.FindByNameAsync(user.UserName);

            return new RegisterResponse { CreatedUserId = getUser.Id, IsSuccessfulRegistration = true };
        }

        Počet odkazů: 2
        public async Task<LoginResponse> Login(UserLoginRequest userLoginRequest)...

        Počet odkazů: 2
        public async Task<ChangePasswordResponse> ChangePassword(UserChangePasswordRequest userChangePasswordRequest, Guid requestingUserId)...
    }
}
```

Obrázek 26 Aplikace - Autentizační služba

Command služby se tedy starají o akce, které nějak ovlivňují naši službu, například registrací uživatelů, vytvářením nových domácností apod. Opět jsou jednotlivé služby rozděleny podle „kategorie“, pod kterou spadají, jako je domácnost, meteostanice nebo uživatel. V těchto

službách probíhá také další ověřování, převážně typu, jestli daný objekt existuje nebo jaké jsou jeho vztahy s dalšími objekty.

Query služby jsou vnitřní strukturou podobné Command službám, ovšem jsou zaměřené na získávání dat. Nemění tedy nijak stav aplikace, ale získávají její současný stav. Většina těchto služeb obsahuje více podobných funkcí, lišících se ve formátu dat, které vrací. Jedny jsou určeny pro získávání všech dat, které odpovídají parametrům. Druhé data vrací po částech, což slouží pro tzv. pagination, tedy stránkování výstupu, kdy se nenačítají všechna data, ale pouze aktuální zvolená stránka o definované velikosti.

7.2.2.2 Objekty a entity

Další součástí Core projektu jsou objekty a modely, používané pro manipulaci dat v rámci aplikace a taktéž jejich odesílání a přijímání z vnějšku serverové části.

Doménové entity jsou interním modelem, který je používaný pro práci s daty uvnitř jednotlivých služeb. V závislosti, jestli jsou data přijímány nebo odesílány, jsou zdrojem dat modely z API části a DTO objektů, případně z Infrastructure části a odpovídajících databázových objektů. Data Transfer Objects slouží, jak už bylo zmíněno, k odesílání a přijímání dat z externího zdroje. Tyto modely již byly několikrát popsány, přičemž se nijak od předchozích variant neliší, proto není nutné je nijak dál rozvádět.

7.2.3 Infrastruktura

Poslední projekt v rámci serverové části aplikace má za úkol definovat třídy a funkce spojené s používanou databází. To zahrnuje i objekty, repozitáře a objekty v rámci Entity frameworku. Je důležité zmínit, že databáze a aplikace obecně pracuje s přístupem Code First, kdy první je vytvořena struktura databázových objektů v rámci projektu a na jejím základě až posléze vytvořena databáze samotná.

7.2.3.1 Databázové objekty

Podobně jako objekty a entity v Core části projektu, i tyto objekty slouží k ukládání dat. Oproti předchozím modelům ovšem do určité míry reprezentují skutečné vztahy, podobu a obsah tabulek v databázi samotné. Jedna ku jedné nejsou z toho důvodu, že kvůli lepší práci s těmito objekty v rámci aplikace obsahují kromě cizích klíčů ve formě GUID i přímou referenci na objekt, která není v databázovém systému přímo vyobrazena. Operace prováděné s těmito modely mají přímí vliv na stav uvnitř databáze jako takové.

7.2.3.2 Repozitáře a databázový kontext

Repozitáře se starají o všechny změny a operace s daty v databázi. Jednotlivé repozitáře mají přístup k připojení do databáze ve formě kontextu, skrz který data získávají, vytváří, mění a také mažou. Funkce, které obsahují, jsou volány většinou ze služeb v Core projektu.

Kontext aplikace má na starosti databázi jako takovou, k čemuž se využívá Entity Framework. Definují se zde parametry tabulek vytvářených v rámci ASP.NET Identity, konfigurace databáze a ostatní tabulky, které bude databáze obsahovat. Lze zde definovat i vztahy těchto tabulek, ovšem základní vztahy lze vytvořit i v rámci modelů samotných pomocí atributů.

```
namespace Tempestas.ServerInfrastructure.Repository
{
    Počet odkazů: 2
    public class ApplicationUserManager : IApplicationUserManager
    {
        private readonly UserManager<UserDo> _userManager;
        private readonly ApplicationDbContext _context;
        private readonly IMapper _mapper;

        Počet odkazů: 0
        public ApplicationUserManager(UserManager<UserDo> userManager, ApplicationDbContext context, IMapper mapper)
        {
            _userManager = userManager;
            _context = context;
            _mapper = mapper;
        }

        Počet odkazů: 2
        public async Task<IdentityResult> AddToRoleAsync(User user, string role)
        {
            var userDo = await _context.Users
                .FirstOrDefaultAsync(x => x.UserName == user.UserName);

            return await _userManager.AddToRoleAsync(userDo, role);
        }

        Počet odkazů: 2
        public async Task<IdentityResult> ChangePasswordAsync(string username, string oldPassword, string newPassword)...

        Počet odkazů: 2
        public async Task<bool> CheckPasswordAsync(string username, string password)...

        Počet odkazů: 2
        public async Task<IdentityResult> CreateAsync(User user, string password)
        {
            var userDo = _mapper.Map<UserDo>(user);

            return await _userManager.CreateAsync(userDo, password);
        }

        Počet odkazů: 4
        public async Task<User> FindByNameAsync(string username)...

        Počet odkazů: 2
        public async Task<IList<string>> GetRolesAsync(User user)...
    }
}
```

Obrázek 27 Aplikace - Správce uživatelů / uživatelský repozitář

7.2.3.3 Migrace

Součástí EF jsou tzv. migrace. Jedná se, zjednodušeně řečeno, o popis změn vzhledu tabulek a databáze v nějaký moment. Při provedení změny v rámci aplikace ve struktuře databáze

jsou tyto změny uloženy do migrace, podle které se při aktualizaci provedou v databázi změny. V případě nutnosti lze pomocí migrací vracet databázi do předchozích stavů.

Současný stav databáze je poznačen v rámci třídy ApplicationDbContextModelSnapshot, která v sobě má popsán současný vzhled databáze. To zahrnuje datové typy, název sloupce, jestli je hodnota generována automaticky, zda se jedná o primární klíč atd.

7.2.4 Sdílená část a další

Tato část je sdílena mezi backendovou částí a frontendovou. I přes to, že by do této sekce mohly být zahrnuty i další komponenty, primárně modely, které jsou v podstatě identické, jsou zde "pouze" definice koncových bodů. Tím jsou myšleny URL adresy jednotlivých stránek, jako je přihlašování nebo zobrazení konkrétních dat. Také se zde nacházejí adresy API koncových bodů, sloužící pro komunikaci uživatele, potažmo klientské části a meteostanice se serverem. Taktéž jsou zde definovány uživatelské role, v našem případě role Uživatel a role Přispěvatel.

Je také vhodné zde zmínit, že většina projektů obsahuje svou variantu třídy Utilities, která obsahuje pár pomocných funkcí. U klientských komponent jsou zde funkce na skládání názvů tříd v rámci class parametru, v případě serverového Core projektu to je třída pro generování náhodného řetězce, použitého pro vytváření hesel u uživatelů používaných v rámci meteostanic. U projektu Infrastructure to jsou zase třídy na kontrolu GPS souřadnic a funkce ošetřující výjimky při přidávání a aktualizování objektů v databázi.

7.2.5 Swagger

Během vývoje byl také hojně využíván nástroj Swagger [73]. Jedná se o sadu nástrojů, určených pro vývoj API, zahrnující možnosti, jako je návrh, sestavení, dokumentace a testování těchto přístupových bodů. [73]

Rozhraní tohoto nástroje je po přidání do projektu přístupné z dedikované adresy v rámci naší aplikace. Zde jsou zobrazeny veškeré koncové body, rozdělené podle jejich typu, nadřazeného kontroleru apod. Jelikož je přístup k většině koncových bodů podmíněn přítomností bearer tokenu v hlavičce dotazu, je zde k dispozici i možnost jeho získání a následného přidání do hlaviček všech následujících dotazů na koncové body.

Jednotlivé koncové body mají také vytvořený jakýsi vzorový příklad, který ukazuje, co daná funkce přijímá za parametry a co na oplátku vrací. Tato funkcionalita je generována

automaticky, je ovšem vždy možné ji přímo v kódu upravit, případně i změnit. Dodatečně také umožňuje popsat tyto API, což může sloužit jako forma dokumentace aplikace jako takové.

V našem případě byl tento nástroj použit hlavně na testování funkčnosti API koncových bodů. Lze jím například izolovat problém v komunikaci mezi serverovou částí a tou klientskou.

7.2.6 Databáze

Finální podobu databáze tvoří celkem deset tabulek. Část z nich jsou tabulky vygenerované pomocí ASP.NET Identity [26], což je systém pro autorizaci a autentizaci, sloužící k usnadnění implementace právě této funkcionality. Generuje si několik tabulek, některé v našem případě nepotřebné, které utvářejí pomyslný základ starající se o přihlašování a s tím spojených funkcí.

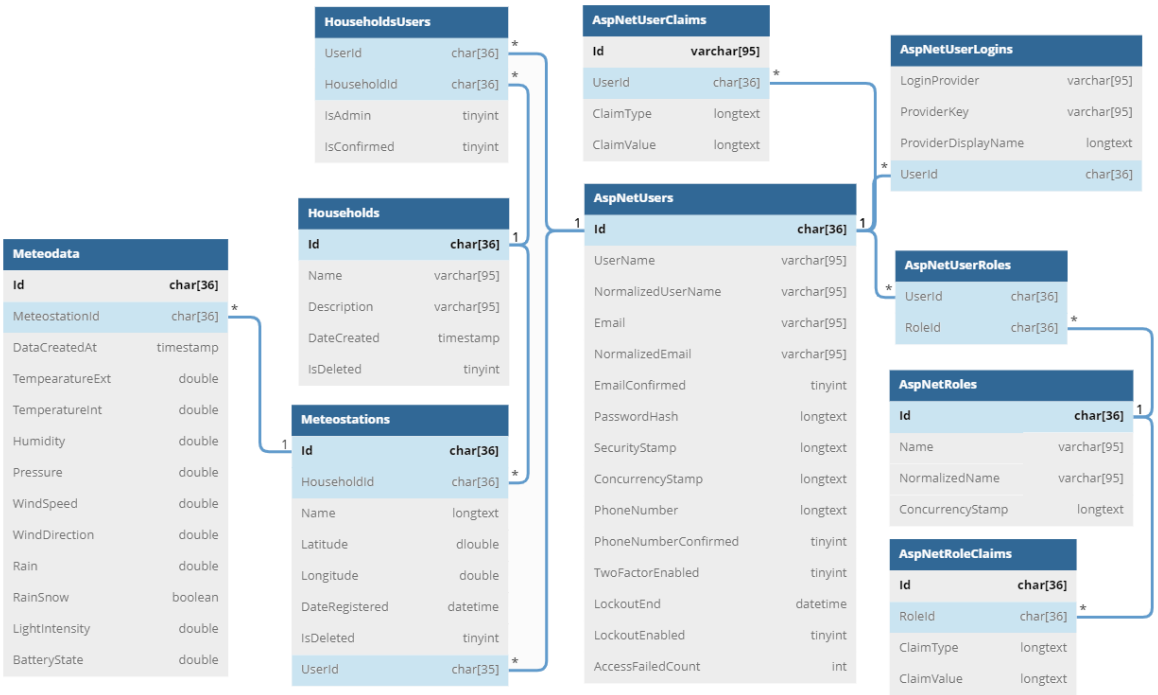
Oproti základní implementaci jsou zde změněny některé věci, týkající se v podstatě pouze unikátního identifikátoru uživatele a všech propojených tabulek. Pro naše potřeby, hlavně z důvodu pohodlí a přehlednosti, je místo výchozího datového typu používán typ GUID, což ovšem přineslo celou řadu problémů. Pro správné propojení tabulek musí mít propojené parametry stejný datový typ, u nás tedy GUID, který je tvořen sérií čísel a písmen v textové podobě, která má pevně stanovenou délku. Kvůli tomu vznikl problém v kompatibilitě s použitým databázovým systémem MySQL. Tento systém se totiž v některých věcech liší, pro nás naneštěstí i ve velikostech některých datových typů. Proto se musí při definici použitého datového typu upravit i velikost, kterou bude zabírat v databázi. Entity Framework, který se v naší implementaci stará o databázi, se ovšem dokáže při správné konfiguraci o tento problém postarat.

Ostatní tabulky jsou přidány „dodatečně“ a definují funkce a strukturu systému v této práci. Ty se během vývoje částečně měnily, především kvůli přehlédnutí některých možností a vzešlé problematice při definování systému.

7.2.6.1 Tabulky

Jako pomyslný střed automaticky vytvořených tabulek je ta starající se o uživatele databáze, pojmenovaná AspNetUsers. Pomocí Id uživatele, uvedeného v této tabulce, je propojena s většinou ostatních tabulek. Kromě přidružených tabulek k autentizaci a autorizaci se konkrétně jedná o propojení uživatele s domácností a definování uživatele, jehož účet slouží

v současné implementaci k vkládání meteorologických dat. Tabulka dále obsahuje hlavně údaje o uživateli, jako je jeho heslo v podobě HASH otisku, uživatelské jméno nebo třeba emailovou adresu či telefonní číslo. Nakonec zahrnuje i proměnné, které jsou součástí případného dalšího zabezpečení nebo k ověření a potvrzení jiných parametrů.



Obrázek 28 Struktura databáze

Další důležitou tabulkou je Households. Ta zahrnuje informace o domácnostech, jako je jméno, případný popis domácnosti a datum vytvoření. Také obsahuje atribut, označující, zda je domácnost aktivní. To slouží jako nahrazení přímého smazání záznamu z databáze, což by mohlo poškodit jiné záznamy, které jsou s domácností propojena.

Členství v domácnosti je vytvořeno pomocí další tabulky, která obsahuje odkaz na domácnost a na uživatele. Toto spojení se také stará o roli uživatele v rámci domácnosti, tedy jestli má správcovská práva a může zasahovat do nastavení domácnosti, a zda je uživatel plnohodnotným členem tím, že přijal pozvánku do dané domácnosti.

K domácnostem jsou následně připojeny meteostanice. Tato tabulka obsahuje především podrobnosti o meteostanici samotné, tedy název a její umístění vyjádřené pomocí souřadnic zeměpisné délky a šířky. Kromě toho jsou zde reference jak na domácnost, do které je meteostanice přiřazena, tak i na uživatele, který slouží jako pomyslný zdroj k ukládání dat. Tento uživatel není přímo členem žádné domácnosti, ovšem data dokáže vkládat jenom

v rámci meteostanice a domácnosti, ke které je přiřazen. Podobně jako u domácností, i tato tabulka má atribut, zda je záznam smazán.

Jako poslední bude zmíněna tabulka s meteorologickými daty. Ta slouží jako reprezentace získaných meteorologických dat, jako je teplota, tlak nebo vlhkost v daný časový okamžik. Mimo tyto parametry jsou v tabulce i údaje o zdroji dat, tedy meteostanici, ze které data pochází a údaj o meteostanici samotné v podobě stavu interní baterie. Tento stav je samozřejmě nepotřebný pro případ, že meteostanice baterie nemá a je napájena jiným zdrojem, než je v našem případě solární panel.

7.3 Meteostanice

Hardware samotné meteostanice byl tvořen průběžně s vývojem softwarové části systému. Jedná se totiž v současné době o jediný způsob, mimo využití nástroje Swagger, jak vkládat do systému data.

Co se odesílání dat do našeho systému týče, ten probíhá v intervalech po deseti minutách. Skutečné rozestupy jsou o něco delší, jednak kvůli tomu, že na začátku každého nového cyklu musí nejdřív dojít k zaznamenání samotných dat, ale i z důvodu, že meteostanici se nemusí povést připojení k internetu a odeslání samotných dat na první pokus. Čas přijetí dat je v současné implementaci zaznamenáván na straně serveru, i když pro implementaci možného „backlogu“ dat by bylo nutné, aby meteostanice samotná znala současný čas, podle kterého by data zaznamenávala a s kterým by je i posílala.

7.3.1 Provedení

Konstrukčně odpovídá meteostanice návrhu, popsaném ve stejnojmenné části této práce. Základem je tedy Arduino Nano 33 IoT, ke kterému jsou připojeny všechny ostatní elektronické části, jako jsou senzory a systém napájení. Konstrukce, která spojuje komponenty dohromady, je vytvořena z hliníkových tyčí 2x2cm různé délky, vzájemně propojených pomocí šroubků a kovových spojek. Komponenty jsou pak k této konstrukci buď přímo přišroubovány, nebo připojeny pomocí malých plastových držáků, vytvořených jako 3D model a následně vytisknuté na 3D tiskárně. Jako materiál na tyto spojovací součásti byl zvolen materiál PETG, který dokáže oproti jiným běžně používaným materiálům používaným pro 3D tisk, lépe vystát teplo a mechanické zatížení.

Meteostanice a její elektrické komponenty jsou napájeny pomocí solárního panelu a dvou lithium iontových baterií. Tím je zaručeno, že energie bude poskytována nepřetržitě, i během špatného počasí a nedostatku slunečního svitu.

7.3.2 Iterace

Při vývoji prošla stanice několika stádii. V první byla umístěna ve vnitřních prostorách, kde k ní byl jednoduchý přístup. Kromě toho byl jako zdroj použit USB kabel připojený přímo k elektrické síti pomocí odpovídajícího zdroje. V dalších etapách poté docházelo k postupnému připojování všech senzorů a sestavování části solárního napájení. Po zapojení všech senzorů byla poté meteostanice umístěna do venkovních prostor, kde je vyobrazena i na následující fotografii. V tento moment dochází stále k napájení pomocí USB kabelu, který zároveň slouží i k nahrávání aktualizovaných verzí programu do mikrokontroleru a postupnému vylepšování funkcí meteostanice samotné. Poslední fáze spočívala v dokončení obvodu pro solární napájení a nahrazení zdroje energie.

V době dokončování této práce meteostanice běží bez vnějších zásahů po dobu zhruba jednoho týdne. Jediným neduhem je nefunkčnost jednoho ze senzorů, konkrétně pro měření intenzity slunečního záření, což je pravděpodobně způsobeno tím, že senzor samotný pracuje s 5 V napájením a logikou, kdežto systém samotný pracuje s logikou 3.3 V. Pro správné fungování by byl nejspíše potřeba dodatečný obvod, starající se o převod mezi oběma úrovněmi napětí.

Obrázek 29 Nasazená meteostanice

7.4 Nasazení

Naše aplikace bude nasazena na vlastním zařízení, konkrétně Raspberry Pi. Jedná se o jednoduché řešení, jak si hostovat nějakou službu pro vlastní potřeby. Samozřejmě existuje možnost profesionálního hostování systému u některé z existujících společností, která se o potřebná nastavení a zabezpečení postará. Za tyto služby se ovšem platí a v našem případě, jelikož se nejedná o žádná kritická data nebo službu, je tedy výhodnější si tyto služby poskytovat sám. Proces samotného nasazení je sice komplikovanější, ovšem poměrně přímočarý.

U vývoje projektu byl využit GitHub repozitář, na kterém je nahráno celé řešení. V případě nové verze, případně nového nasazení, lze pomocí příkazu „git pull“ stáhnout aktuální řešení na požadované zařízení. Po stažení je dalším krokem sestavení aplikace. K tomu, a následnému spuštění aplikace, je nutné mít nainstalovány potřebné verze .NET SDK, které aplikace pro svůj provoz potřebuje. Pokud je tato podmínka splněna, lze aplikaci sestavit příkazem „dotnet publish“. Po sestavení už stačí aplikaci nastavit doplněním údajů do konfiguračního soboru a následně spustit.

Mimo samotnou službu na tomto zařízení zároveň běží i databázový systém, který systém používá pro ukládání dat. Ten je také nutné nejdříve stáhnout a naistalovat, ovšem po provedení základního nastavení se už o tuto část nemusíme nijak více starat.

7.5 Nastavení

Žádný systém není stejný a každý potřebuje před nasazením nastavit tak, aby fungoval správným a žádoucím způsobem. Některá nastavení ovšem mohou narušit bezpečnost uživatelovi lokální sítě a poskytnout potencionálním útočníkům příležitost k napadení. Je proto důležité těmto zranitelnostem předcházet.

7.5.1 Server

Pro správné fungování aplikace je po jejím nasazení nutné zadat některé údaje, které jsou unikátní pro každého uživatele. Všechna nastavení se provádějí v rámci souboru appconfig v projektu Server, konkrétně údaje o připojení k databázi a cesta k SSL certifikátu používanému v rámci připojení pomocí HTTPS. Ostatní údaje jsou již předvyplněny, ovšem je doporučeno některé změnit, což se týká především symetrického šifrovacího klíče, používaného pro vytváření JWT tokenů a jejich ověřování.

Při následném spouštění aplikace je navíc důležité specifikovat, které porty má aplikace pro komunikaci používat, přičemž by se měly shodovat s porty, nastavené v rámci směrování portů, popsaném dále.

7.5.2 Databáze

Databázový systém pracuje nezávisle na našem systému na stejném zařízení. Konkrétně byl zvolen systém MariaDB [74], což je relační open source databáze, vycházející ze základu MySQL.

Před jeho použitím je potřebné ho nastavit tak, aby umožňoval přístup naší aplikaci k datům v databázi uloženým. Pro tyto účely je vytvořen účet, potažmo uživatel, který má práva na to pracovat v rámci části databáze určené pro naši aplikaci, ovšem bez práv na zasahování do případných jiných instancí a databází.

Taktéž je nutné vytvořit samotnou databázi. Entity framework, který je součástí naší aplikace, sice dokáže vytvářet databáze samotné, ovšem k tomu je potřeba účet s potřebnými právy. Ty sice lze účtu, který budeme používat pro práci s databází, poskytnout, ovšem představuje to možné bezpečnostní riziko pro ostatní databáze v databázovém systému, četně naší databáze samotné v případě, že přihlašovací údaje k tomuto systému uniknou.

7.5.3 Přístup k zařízení z internetu

Pro zlepšení dostupnosti služby je žádoucí, aby byla dostupná odkudkoliv, proto se součástí systému možnost přistupování k němu i mimo lokální síť, tedy přes internet. K dosažení tohoto cíle je nutné změnit pár nastavení v rámci domácí sítě uživatele.

7.5.3.1 Port forwarding

Pro umožnění přístupu ke zdroji uvnitř sítě je možné využít tzv. směrování portů. To spočívá v tom, že příchozí komunikaci, která je adresována na námi zvolený port, je možné přesměrovat na zařízení se specifikovanou lokální adresu uvnitř naší sítě.

Toto nastavení se provádí v rámci modemu, potažmo routeru v naší síti. Většina těchto zařízení má v nastavení možnost povolení směrování portů, četně definování jednotlivých parametrů. V momentě, kdy se z internetu pokusíme dostat na IP adresu routeru s určeným portem, bude tato komunikace přesměrována na lokální zařízení se serverem na daném portu.

Kromě tohoto nastavení je také vhodné nastavit zařízení, na kterém běží naše aplikace, statickou IP adresu, což zapříčiní, že se nebude v průběhu času měnit. To by mohlo zapříčinit to, že adresa použitá v nastavení směrování portů může přiřadit jinému zařízení, přičemž by komunikace s aplikací nebyla možná.

7.5.3.2 Dynamic DNS

Po konfiguraci směrování portů se sice lze dostat k naší službě z internetu, ovšem pouze pomocí zadání přímo IP adresy, která není formátem příliš uživatelsky přívětivá. Kromě nákupu vlastní domény, tedy webové adresy, přes kterou se můžeme do systému dostat,

existuje možnost použití služby, tzv. Dynamic DNS. Domain Name Systém je, jednoduše řečeno, systém sloužící k převodu IP adres na doménová jména a naopak v rámci internetu jako takového.

Konkrétním řešením, používaným v rámci této práce, je DuckDNS [75]. Po registraci a přihlášení do této služby máme možnost vytvořit si vlastní adresu, potažmo subdoménu, jelikož název stále obsahuje název DuckDNS, kterou poté bude možné používat pro přístup k aplikaci. K té je připojena naše IP adresa, na kterou budou požadavky na vytvořenou adresu přesměrovávány.

Dodatečně lze pomocí programu na našem zařízení pravidelně aktualizovat naši adresu v této službě, jelikož jen málo lidí má přiřazenou externí statickou IP adresu. Toto ovšem není klíčová součást, jelikož adres typu IPv4 je už k dispozici jen velmi málo a šance, že dojde k výměně uživatelovi adresy je velmi malá.

7.6 Funkce systému

Primárním zaměřením systému je sběr meteorologických dat a jejich prezentace uživateli. Uživatelské rozhraní je proto hlavně zaměřeno na zobrazování dat. To získává potřebné informace pro své fungování ze serverové části pomocí REST API, jak už bylo vícekrát zmíněno.

7.6.1 Přehled, registrace a přihlášení

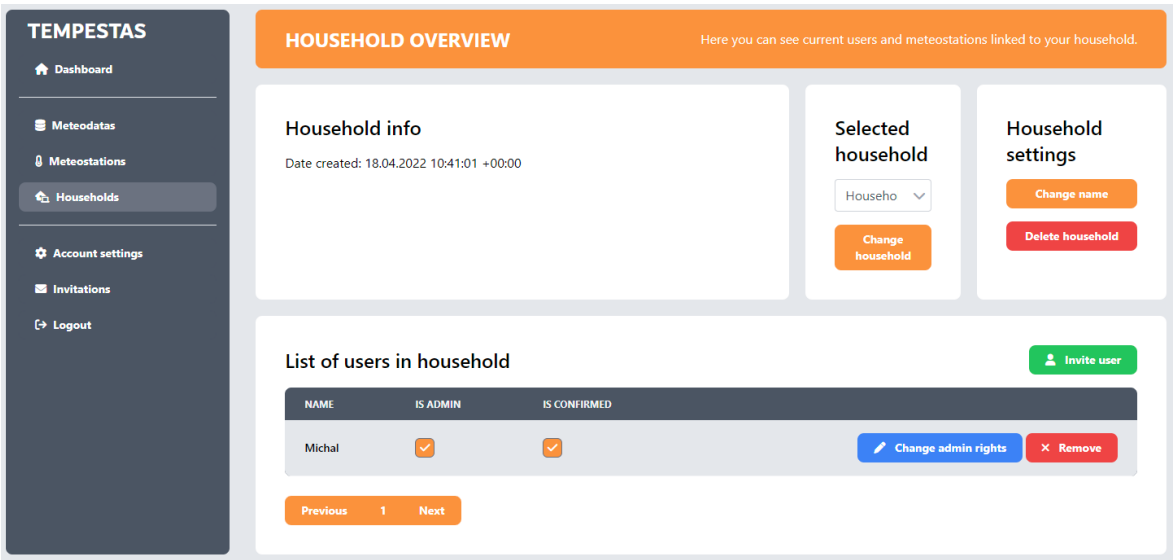
V základu umožňuje systém zobrazovat pouze nejnovější data, která jsou nejblíže fyzické lokalitě přistupujícího uživatele. K získávání geografické pozice slouží integrovaná funkcionalita většiny webových prohlížečů, načež získané souřadnice jsou použity k nalezení nejbližší meteostanice, samozřejmě za předpokladu, že systém má více než jednu meteostanici. K přístupu k dalším funkcím se musí uživatel registrovat a být přihlášen do systému. Registrovat se může přes příslušný formulář, přístupný z úvodní stránky.

Přihlášený uživatel má v sekci Nastavení účtu k dispozici možnosti, jako je změna hesla, vytvoření či smazání domácností, ve kterých je správce, a také smazaní účtu samotného.

Obrázek 30 Aplikace - úvodní obrazovka

7.6.2 Domácnost

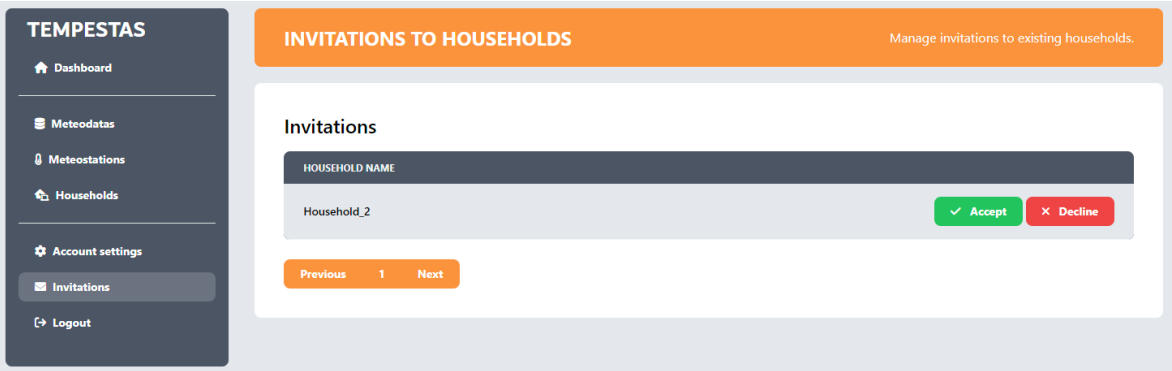
Pokud už je členem nějaké domácnosti, může si prohlížet historická data ze všech meteostanic, které jsou součástí domácnosti. Současně může být členem více domácností, může si tedy volit i „aktivní“ domácnost, v rámci které si data prohlíží. Pokud má zároveň uživatel správcovskou roli v nějaké domácnosti, má přístup k dalším funkcím, mezi kterými je možnost přizvat do domácnosti nové uživatele, přidávat nové zdroje dat atd.



Obrázek 31 Aplikace - okno domácnost

Pokud uživatel není členem žádné domácnosti, může být buď přizván do již existující, nebo může vytvořit domácnost novou, kde bude automaticky správcem. Pro přizvání do domácnosti musí uživatel, který je členem dané domácnosti, a který je správcem, poslat pozvánku.

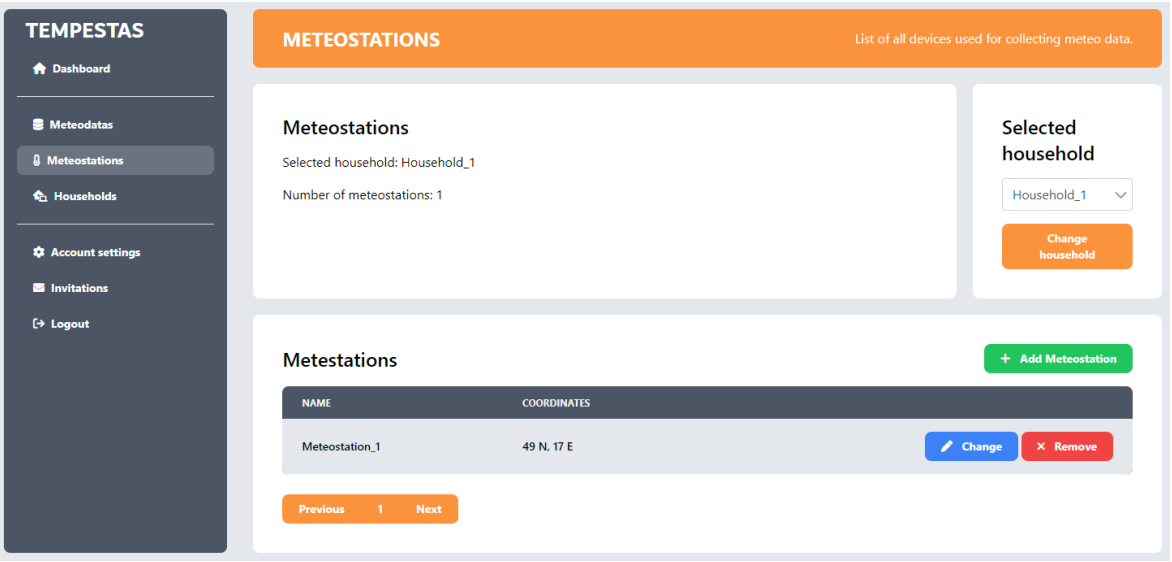
Ta se zobrazí v sekci Invitations, přičemž uživatel může pozvánku buď přijmout nebo odmítnout.



Obrázek 32 Aplikace - okno pozvánky

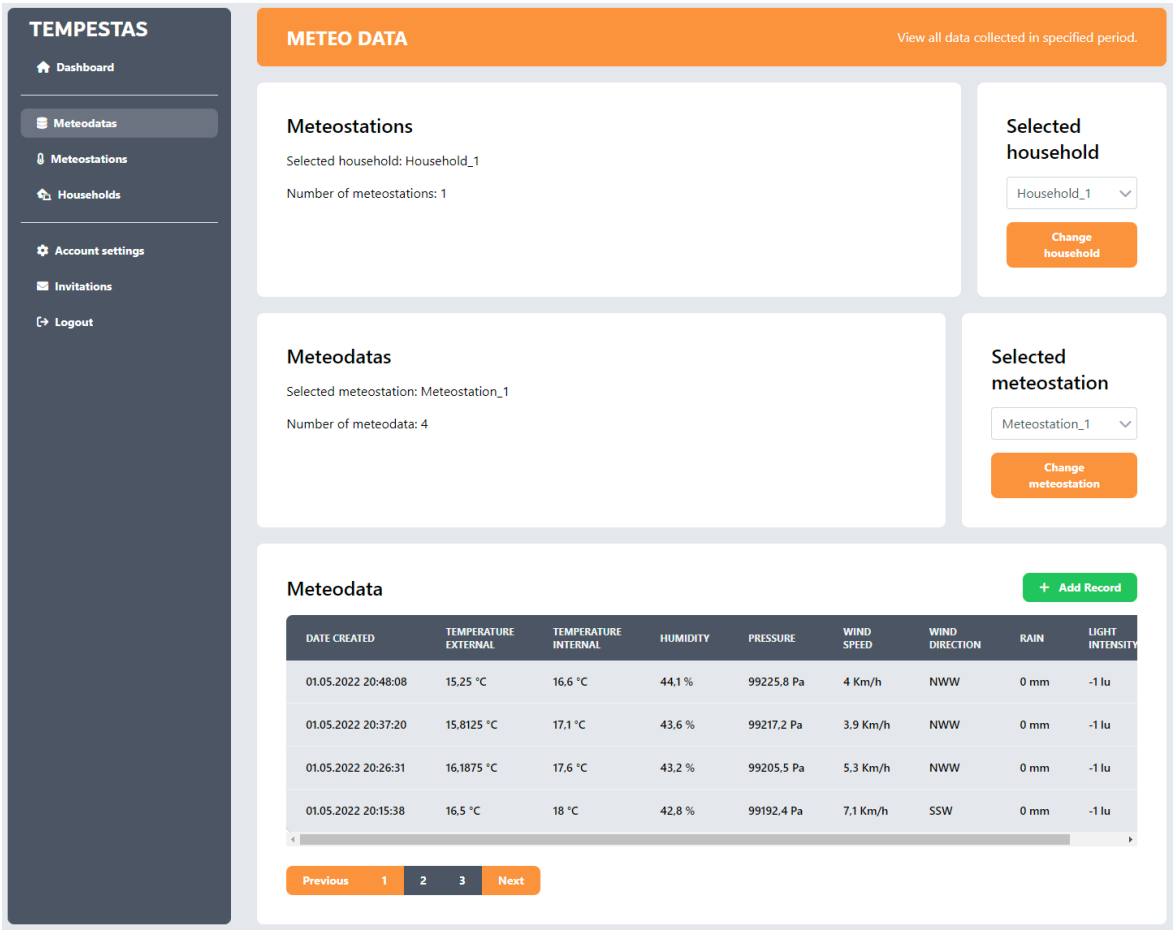
7.6.3 Meteostanice a meteorologická data

V rámci domácnosti lze již pracovat přímo s meteostanicemi. Ve stejnojmenné záložce je umožněno uživateli vytvořit novou meteostanici se základním nastavením. Po inicializaci je k dispozici nakonfigurovat meteostanici, což zahrnuje vyplnění přístupových údajů k Wi-Fi síti, ke které bude konkrétní meteostanice připojena. Výsledkem je soubor, obsahující kód s konfigurací, který stačí pouze nahrát do meteostanice samotné. V současné podobě lze takto konfigurovat pouze jedno konkrétní řešení, ovšem vždy je možnost přidat do systému další varianty, případně, pokud je uživatel více technologicky zaměřen, si může poskytnutý kód upravit sám pro své potřeby.



Obrázek 33 Aplikace - okno meteostanice

Meteorologická data jsou zobrazována v podobě tabulky s hodnotami, seřazenými od nejnovějších po nejstarší. Jednotlivé parametry jsou zobrazeny v upravené formě, u většiny je pouze přidána jednotka, u zbytku je hodnota změněna na člověkem čitelný formát. Například u směru větru je hodnota ve stupních převedena na světové strany.



Obrázek 34 Aplikace - okno meteorologická data

7.7 Zabezpečení

Každá aplikace by měla mít nějakou formu zabezpečení, ať už se jedná o jednoduchou statickou stránku nebo o komplexní systém. Tento systém samozřejmě není jiný.

Zde se zabezpečení dá rozdělit dvou částí nebo kategorií. První je zabezpečení komunikace, týkající se předávání informací mezi klientem a serverem, přičemž druhá část je zaměřena na zabezpečení aplikace jako takové.

7.7.1 HTTPS / SSL

Pro zabezpečení komunikace samotné je provedeno pomocí protokolu HTTPS, který využívá SSL certifikátu. V případě, že by pro hostování aplikace bylo využito služby na toto zaměřené, tuto záležitost není většinou potřeba řešit. Totéž se dá říct v případě, kdyby naše služba nebyla dostupná z internetu samotného.

Certifikát se dá získat i bezplatně na určitou dobu, většinou tři měsíce. V našem případě bylo využito služby ZeroSSL [76], která tyto certifikáty poskytuje. Po vytvoření účtu je zvolena webová adresa, pro kterou chceme certifikát vystavit. V dalším kroku je nutné dokázat, že webový server, na který adresa ukazuje, opravdu náleží nám. K tomu lze použít několik poskytnutých metod, námi zvolenou je ovšem ta, která zahrnuje nahrání poskytnutého souboru do kořenového adresáře klientské části. Tato služba se poté pokusí přistoupit k tomuto souboru a pokud se shoduje s tím, který nám byl poskytnut, bude nám vystaven SSL certifikát.

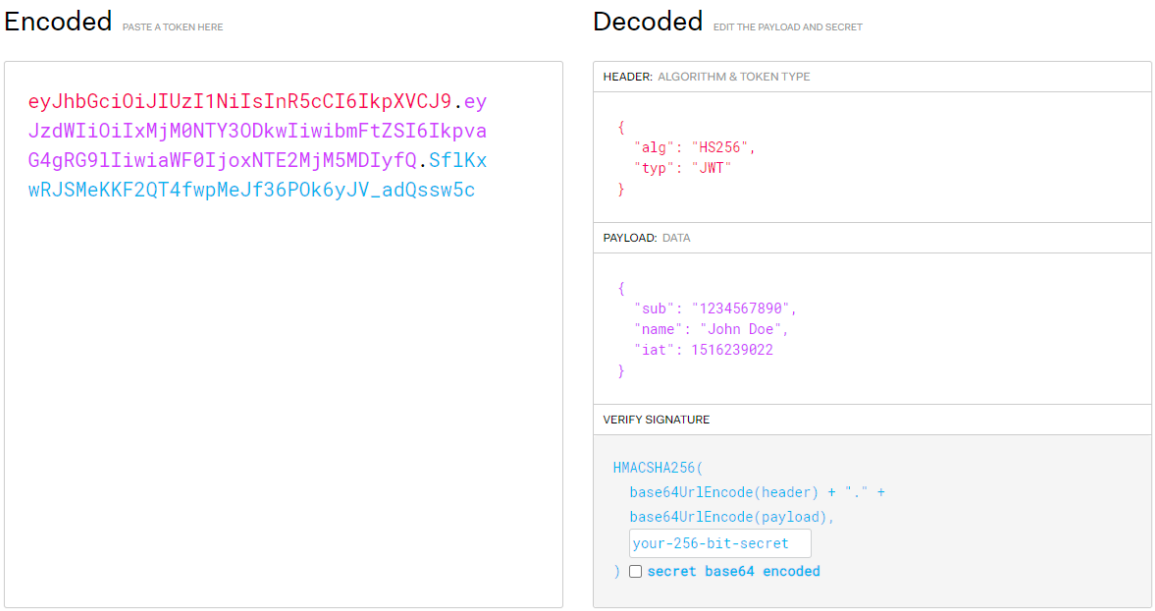
Poskytnuté soubory je nutné před použitím upravit, konkrétně převést na formát použitelný v naší aplikaci, například pomocí kryptografického nástroje, jako je OpenSSL [77]. Cestu k výsledným souborům lze nakonec poskytnout aplikaci samotné.

7.7.2 JWT Token

JSON Web tokeny jsou otevřený průmyslový standard, sloužící pro reprezentaci rolí nebo nároků mezi dvěma stranami [78]. Token samotný se skládá z hlavičky, která obsahuje typ algoritmu a použitého tokenu, těla, které může obsahovat libovolné údaje, převážně tedy role nějakého uživatele, a nakonec z podpisu, který slouží k ověření pravosti tokenu.

V našem systému slouží tento token pro autorizaci operací se systémem. To se týká čtení dat, jejich zapisování, měnění a také mazání. Pro získání tohoto tokenu se musí uživatel přihlásit pomocí svých platných údajů, za což dostane od serveru odpověď, zahrnující i tento token, který vkládá do další komunikace se serverem.

Tyto tokeny se v tomto systému používají i v rámci přidávání dat ze strany meteostanic, jelikož, jak už bylo zmíněno, každá meteostanice má přiřazeného uživatele, který je použit pro vkládání získaných dat. V budoucnu by tento přístup mohl být nahrazen pouze identifikačním tokenem, který by obcházel potřebu uživatele pro přidávání dat.



Obrázek 35 Dekodér JWT tokenů [78]

7.7.3 Uživatelská hesla

Nejdůležitější informací, kterou uživatel ukládá do tohoto systému, jsou hesla. Pro posílení bezpečnosti jsou proto hesla jednotlivých uživatelů ukládána ne v jejich skutečné textové podobě, ale jako tzv. HASH. Použitý algoritmus na základě vstupního textu vytvoří výstup o vždy stejné délce, který je unikátní právě pro tento text a je téměř nemožné z něho zpětně získat text původní.

Důvodem využívání tohoto přístupu na ukládání hesel je hlavně bezpečnost. Pokud by například došlo k úniku databáze s uživatelskými účty, budou místo hesel spojených s konkrétními uživateli dostupné pouze jejich otisky, které nejsou nijak použitelné v rámci jiných systému a aplikací.

V případě ASP.NET Identity, která se stará o přihlašování a správu uživatelů v tomto systému, je využívána na hašování funkce HMACSHA512 [79]. Kromě použití algoritmu samotného je navíc během samotného procesu přidávána i tzv. sůl, tedy série pseudonáhodných bitů, které zajistí, že v případě vytváření haše stejného hesla budou oba výstupy odlišné.

7.7.4 Databáze

7.7.4.1 SQL injection

Jedním z nejběžnějších způsobů, jak útočit na databáze, je tzv. SQL injection. Princip tohoto útoku spočívá ve vložení nežádoucího SQL příkazu přes textový nebo jiný vstup, například uživatelské jméno, který následně bude bez našeho vědomí spuštěn a proveden. Výsledkem můžou být nechtěné změny v databázi či přímo odcizení citlivých údajů. Je proto nutné, aby byly vstupy kontrolovány a přístup k databázi prováděn co nejbezpečnějším způsobem.

Výhodou využití Entity frameworku v této práci je již částečné ošetření tohoto a dalších možných útoků. Framework totiž poskytuje uživateli objekt, pomocí kterého může program přistupovat k databázi a podobu samotných SQL příkazů si vytváří a ošetřuje sám. Samozřejmě pořád poskytuje možnost vytvářet si příkazy vlastní, ovšem tento přístup není doporučován.

Další pomyslnou ochranou je použití LINQ. Ten totiž vytváří dotazy do kolekcí jiným způsobem, než je běžná manipulace s textovými řetězci, čímž je proti tradičním SQL injection útokům imunní. [80]

Jako poslední část ochrany proti tomuto útoku lze považovat i samotné API koncové body. Každý z nich je totiž zaměřen na konkrétní typ dat a přijímá pouze parametry s tím spjaté. Neexistují proto žádné univerzální přístupové body, přes které by šlo získat v podstatě cokoliv. Účelem tedy není přímo zabránění, ale spíše ztížení případných útoků a napadení systému.

7.7.4.2 Přístup do databáze

Do databázového systému se přistupuje pomocí speciálně vytvořeného účtu. Ten má pouze omezená práva na to, co může v databázi vykonávat za akce. Hlavním omezením je možnost měnit data pouze v rámci databáze určené tomuto systému, přičemž ostatní databáze nemůže nijak ovlivnit, například zápisem dat či smazáním. V současné verzi tohoto systému nemá tento uživatel žádná další omezení, co se týká přiřazené databáze. Může v ní tedy vytvářet, upravovat a mazat data i tabulky samotné. Pokud by se měli jeho práva omezit, mířilo by se hlavně na možnost měnit strukturu databáze, myšleno tabulky a jejich spojení. Práva na operace s daty musí zůstat pro správné fungování systému.

Co se týče přímo přístupových údajů, ty jsou jako Environment variables umístěny v konfiguračním souboru appsettings jako prostý text. To nemusí být ideální, hlavně kvůli

možnému odcizení a zneužití těchto údajů. Existují sice alternativy určené právě na ukládání citlivých údajů, konkrétně například nástroj Secret Manager, u kterého jsou ale bohužel, stejně jako u předchozí varianty, data opět v nezašifrované podobě. Je proto vhodným řešením kombinovat zmíněné způsoby uložení s dalším typem zabezpečení, nejlépe nějakou formou šifrování.

7.7.5 Server

Poslední klíčovou částí zabezpečení je zamezení přístupu na zařízení, na kterém tento systém běží. V případě, že by neoprávněná osoba získala přístup k zařízení, lze některá existující zabezpečení lehce obejít a dostat se tak ke klíčovým částem systému a k citlivým datům. Je proto nutné dbát na správné nastavení přístupů a jednotlivých služeb.

V našem případě je ke správě zařízení, konkrétně Raspberry Pi, používán systém vzdálené správy. Ten umožňuje ovládat zařízení přes internet stejným způsobem, jako kdyby pracoval uživatel přímo s fyzickým zařízením. Tím, že je tento přístup dostupný z jiných zařízení, zde existuje možná zranitelnost, přes kterou by bylo možné zařízení a služby na něm kompromitovat. Hlavní je proto co nejvíce omezit tyto přístupy, například povolením přístupu pouze specifikovaným zařízením a nastavením co nejsilnějších hesel, pomocí kterých systém spravujeme.

Kromě zabezpečení zařízení samotného je nutné hlídat i přístupy k jednotlivým službám, které na něm běží. To je konkrétně případ samotného systému v této práci, ke kterému se lze dostat i z internetu, přičemž lze toto spojení zneužít k získání přístupu k dalším částem zařízení. Tento problém se také týká databázového systému, který má v základu vytvořen účet s nejvyššími možnými právy, které mohou být zneužity pro odcizení citlivých dat nebo jejich poškození. Klíčové je zde proto omezení možných zranitelností, jak už limitováním přístupu k některým zdrojům, tak i deaktivací výchozích a nezabezpečených účtů.

8 REÁLNÝ PROVOZ

Před vznikem této práce existoval prototyp systému, zde nazvaný jako Proof of concept. Ten, společně se samotným systémem, fungoval po určitou dobu a sbíral reálná data, na základě, kterých posléze vznikaly další úpravy a poznatky.

8.1 Proof of concept

Zhruba dva roky zpátky byl vytvořen systém, který pracoval s podobnou myšlenkou, jako obsahuje tato práce. Jednalo se pouze o jednu neúplnou meteostanici, která přímo ukládala data na lokální server, bez rozdělení do domácností, bez uživatelů a jejich ověřování a bez možnosti přístupu z internetu. Systém se skládal ze základních senzorů na měření teploty a tlaku, napájen byl přes USB kabel připojený přes zdroj přímo do elektrické sítě. Server byl v základu v podstatě totožný, používal stejný způsob nasazení aplikace i stejnou databázi.

Po vytvoření návrhu této diplomové práce posloužil tento původní projekt, četně zhruba měsíce dat, jako jakési „ověření konceptu“, které potvrdí možnost realizovatelnosti projektu. Samozřejmě existují i jiné podobné systémy, které by potvrdili proveditelnost, ale jako základ posloužil tento systém dobře.

Podle dat, které systém nasbíral vyplívá, že pokud je systém v uzavřené krabici vystaven přímému slunečnímu záření, dochází k výkyvům teplot, které zkreslují skutečnou hodnotu teploty. Toto jenom potvrzuje nutnost přítomnosti externího senzoru teploty, krytého radiačním štítem a umístěním i ostatních senzorů mimo přímí sluneční svit.

Tento koncept také dobře posloužil k návrhu řešeného systému, hlavně co se funkcionalit týče. Taktéž bylo během jeho tvorby překonáno pár překážek, které by se jinak musely řešit až při tvorbě finálního systému.

8.2 Současné řešení

Během vývoje finálního systému byl také aktivně sestavován hardware pro meteostanici samotnou. Po dokončení aplikace a jím nasazení byl systém funkční, i když hardware samotné meteostanice nebyl kompletně dokončen kvůli nedostupnosti některých součástek a nefunkčnosti originálního návrhu a části následných iterací. Systém celkový byl ovšem uveden do takového stavu, kdy dokáže z velké části fungovat a plnit svůj účel.

Co se týká dat získaných během pár dnů útržkovitého provozu lze říct, že oproti předešlé verzi došlo ke zlepšení v několika ohledech. Prvně se jedná o teploty, které jsou díky použití

radiačního štítu stabilnější a s menšími výkyvy než pouze uzavřený v neprodyšné krabici. Také byly připojeny další senzory, které nebyly součástí předchozího prototypu.

Během testování vyvstalo pár nedostatků, které šlo jen těžko předvídat. První se týká napájení samotného mikrokontroleru a senzorů k němu připojených. Původně bylo v plánu využít solární napájení, jehož výstupní napětí bude převedeno na 3.3 V a použito jako vstup pro mikrokontroler. Během sestavování ovšem vyšlo najevo, že i přes některé komentáře na internetu nelze tato úroveň napětí použít. Proto byla vyzkoušena řada alternativních řešení, u které většina neprodukovala dostatečně velký proud na pokrytí výkyvů v proudu, které produkuje Wi-Fi modul, proto komunikace nemohla pracovat správným způsobem.

Ve výsledném řešení se již podařilo zprovoznit meteostanici tak, aby fungovala i bez napájení přímo ze sítě. To je i přes napájení pomocí solárního panelu a baterií stabilní a dostatečné pro delší provoz meteostanice. Problémovým se jeví senzor určený na měření intenzity slunečního záření, který po pár měřeních přestane správně pracovat a je nutné ho resetovat. Meteostanice samotná také občas neodešle data, přičemž správný provoz se může obnovit až po pár hodinách, jestli vůbec. Na řešení tohoto problému by bylo nutné více času a iterací softwaru na ovládání meteostanice jako takové.

9 MOŽNÁ VYLEPŠENÍ

Během tvorby systému vyvstalo spousta aspektů a funkcí, které by bylo možné v budoucnu rozšířit nebo vylepšit. Pokud by tedy vznikla další verze tohoto systému, pravděpodobně by zahrnovala většinu z dále navrhnutých vylepšení a změn v implementaci. Většina návrhů zde vznikla buď jako reakce na vyvstávající problémy, nebo jako funkcionalita, na jejichž implementaci nezbyl během vývoje čas.

9.1 Změny v implementaci

Změna, která by byla asi nejužitečnější z hlediska použitelnosti, je implementace samotného backendového API. Hlavním cílem by bylo zjednodušit získávání dat pomocí filtrování nebo specifikování požadavků. V současnosti se musí pro každý atribut, na jehož základě chceme filtrovat, přidat zvlášť jako nový koncový bod. Tato změna by zjednodušila dotazování jako takové a také umožnila jednodušší rozrůstání aplikace do budoucna. Samozřejmě by zde vznikly nové problémy, hlavně s ošetřením a sanitací vstupů, aby nedošlo k neúmyslnému nebo i úmyslnému narušení systému a dat.

Taktéž by bylo vhodné předělat způsob ověřování meteostanic. Ty momentálně fungují jako uživatel se speciálními právy, což nemusí být dlouhodobě vhodné řešení. Proto by nebylo od věci změnit tento systém, pravděpodobně na takový zahrnující nějakou formu klíče, který by byl uložen pouze na meteostanici a nikdo jiný by k němu neměl přístup. Nejlepší variantou by bylo využití integrovaného krypto čipu na zvoleném mikrokontroleru Arduino Nano a ke komunikaci použít kombinaci symetrické a asymetrické kryptografie.

Jako poslední změnu zde zmíním oddělení klientské části od té serverové. I když se virtuálně jedná o jeden celek, serverová část může pracovat nezávisle na té klientské. Toto by bylo vhodné pro uživatele, kteří mají zájem pouze o část serverovou, případně si chtějí vytvořit vlastní implementaci webového rozhraní. Současné spojení bylo zvoleno hlavně proto, aby byly obě části pohromadě v jednou projektu a tím se usnadnil vývoj jako takový.

9.2 Další funkcionalita

9.2.1 Alternativní zdroje

Jako hlavní dodatečnou funkcionalitu lze považovat rozšíření o možnost získávat data z alternativních zdrojů. Tímto je myšleno například webové služby s živými daty, které využívají pro získávání dat podobný systém jako tato aplikace. V podstatě by se jednalo o možnost

zvolit si typ vstupního zařízení nebo služby, přičemž by se následně server samotný staral o posílání požadavků na tyto služby, místo aby na data sám čekal. Většina služeb ovšem vyžaduje nějaký typ autorizace / autentizace, proto by bylo nutné vytvořit další část aplikace, která by se starala o propojování účtů, vkládání autorizačních tokenů a podobně.

9.2.2 Přihlašování přes externí služby

Dále by se mohlo jednat o možnost přihlašování pomocí externích služeb, jako je například Google nebo Facebook. Tím by odpadala nutnost vyplňování údajů a uživatel by se registroval a následně i přihlašovat v podstatě jedním klikem.

9.2.3 Propojení domácností

Jedním z návrhů, které nebyl realizován, byla možnost propojovat domácnosti mezi sebou. Místo toho, aby uživatelé musely být členem více domácností, aby měly přístup k dalším zdrojům a údajům, by byly propojeny přímo domácnosti. Tím by odpadaly další úkony na straně běžných uživatelů, jelikož o toto spojení by se starali správci domácností.

9.2.4 Prezentace dat

Co se týká prezentace dat, v současné podobě systému chybí komplexnější zobrazení statistik a dat samotných. To je zapříčiněno jednak nedostatkem času pro potřebnou implementace, ale také nedostatkem použitelných dat. I přes pár stovek záznamů je jich příliš málo na to, aby bylo možné odpovídající funkce pořádně implementovat a otestovat.

Konkrétně by se mohlo jednat o různorodé grafy, které by ukazovaly průběh hodnot během dne, například vývoj teploty. Kromě denních shrnutí by se mohlo jednat i o měsíční a roční přehledy nebo o jakési rekapitulace, ukazující například, jaká byla před rokem v tento den teplota.

9.2.5 Jazykové nastavení

Jednou z nevýhod současné verze je její lokalizace do angličtiny. To může pro některé potencionální uživatele představovat překážku, týkající se jejich schopnosti pracovat se systémem. Toto se netýká ani tak zobrazování dat, jelikož význam se dá mnohdy odvodit například z jednotek, jako spíše dalších funkcionalit, nastavování a práce se systémem samotným.

.NET podporuje tvorbu souborů, sloužících pro tvorbu lokalizace. Nazývají se resources, a mají podobu jednoduchých tabulek se sloupci se jménem zdroje, jeho hodnotou a případným

komentářem. Na tyto soubory se po jejich vytvoření odkazuje na místě, kde jsou potřeba a následná volba jazyka je buďto závislá na jazyku prohlížeče, nebo na vlastní volbě, pokud je tato funkcionalita implementována.

9.2.6 Alternativní zobrazení dat

Momentálně je jediným způsobem, jak zobrazit uložená data v uživatelsky přívětivé podobě, webová aplikace. Jelikož ta ke získání informací využívá serverovou část, která funguje na základě API přístupových bodů, může být pro čtení, vytváření, upravování, či mazání dat, použita jakákoliv jiná vhodná implementace. Jako jedna z možností by mohla být aplikace pro mobilní telefony, která bude zastávat roli klientské části dostupné z webu.

Aplikace samotná může mít všechny, nebo pouze některé, funkcionality. Pokud budeme chtít mít aplikaci pouze jako zobrazovací zařízení, nemusí zahrnovat vytváření domácností ani přidávání nových zdrojů dat.

Podobným způsobem by mohlo pracovat i zařazení, které by zobrazovalo základní údaje z nejbližší stanice, podobně jako u komerciálních zařízení. Displej by mohl využívat technologie E-Ink, která spotřebovává energii pouze na překreslování zobrazených dat. V kombinaci se zapínáním Wi-Fi modulu pro získávání dat, by mělo zařízení minimální spotřebu a mohlo by fungovat bez nabíjení nebo přímého zapojení do elektrické sítě i několik měsíců.

9.2.7 Nasazení v Docker kontejneru

Pro zjednodušení nasazení aplikace a její konfigurace existuje možnost tvorby tzv. Docker kontejneru [T]. Jedná se o způsob běhu aplikace ve svém vlastním prostředí, místo toho, aby program běžel přímo v rámci operačního systému. Princip lze přirovnat k sandboxovém prostředí, na kterých fungují dnešní mobilní operační systémy, nebo k virtuálním strojům v rámci počítačových a serverových platforem. Toto by umožňovalo zahrnout veškerou konfiguraci a potřebné nástroje, knihovny a zdroje přímo v rámci jednoho balíčku. Uživatel by posléze stáhnul tento kontejner a který by potom bez dalšího nastavování spustil.

9.2.8 Integrace s chytrou domácností

Jako poslední možné vylepšení byla zvolena integrace s chytrou domácností, jako je například HomeKit a podobné. Ta slouží nejen k ovládání částí domácnosti, jako jsou například světla nebo rolety, ale i pro připojení různých senzorů a možnosti získaná data prohlížet. Kromě senzorů pro monitorování kvality ovzduší a systémů pro zabezpečení domácnosti je

možné k chytré domácnosti připojit i teploměry, a i přímo meteostanice. Meteostanice by tedy byla buď přímo připojena do tohoto systému nebo by data poskytovala skrz náš systém samotný. Chytrá domácnost by pak následně mohla reagovat na venkovní počasí, například zapnutím topení, případně klimatizace, když by teploty dosáhly nastavené hranice.

ZÁVĚR

Cílem této práce bylo vytvořit systém, který by umožňovat získávat, zpracovávat a zobrazovat meteorologická data a spravovat jejich zdroje, v tomto případě meteostanice.

V teoretické části byl popsán hardware a software použitý ke tvorbě systému, četně některých alternativ, technologie a návrhové vzory, které byly využity ke tvorbě systému, a základní popis meteorologie jako takové.

Praktické část zahrnovala proces vývoje a návrh systému, popis implementace, reálný provoz systému, a nakonec i možná vylepšení a změny funkcionalit.

Systém samotný je během dokončování této práce stále funkční a přístupný. Také je v plánu aplikaci a meteostanici samotnou dále rozšiřovat a vyvíjet, přičemž hlavními požadavky budou hlavně osobní potřeby autora.

SEZNAM POUŽITÉ LITERATURY

[1] KERNIGHAN, Brian W. a Dennis M. RITCHIE. Programovací jazyk C. Brno: Computer Press, 2006, 286 s. ISBN 802510897X.

[2] C Language Introduction. Geeks for Geeks [online]. Noida, Uttar Pradesh, Indie: GeeksforGeeks, 2021 [cit. 2022-04-16]. Dostupné z: https://www.geeksforgeeks.org/c-language-set-1-introduction/

[3] 2021 Developer Survey. Stack Overflow [online]. New York, NY, USA: Stack Exchange, 2022, 2022 [cit. 2022-04-16]. Dostupné z: https://insights.stackoverflow.com/survey/2021

[4] Introduction to C++ Programming Language. Geeks for Geeks [online]. Noida, Uttar Pradesh, Indie: GeeksforGeeks, 2021 [cit. 2022-04-16]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-c-programming-language/

[5] CHAND, Mahesh. What is C#. C# Corner [online]. Philadelphia, PA, USA: CSharp, 2020 [cit. 2022-04-16]. Dostupné z: https://www.c-sharpcorner.com/article/what-is-c-sharp/

[6] How is C# compiled?. Manning free content center [online]. San Francisco, CA, USA: Automattic, 2020 [cit. 2022-04-16]. Dostupné z: https://freecontent.manning.com/how-is-c-compiled/

[7] JavaScript - Overview. Tutorials point [online]. Raleigh, NC, USA: Tutorials Point, 2022 [cit. 2022-04-16]. Dostupné z: https://www.tutorialspoint.com/javascript/javascript_overview.htm

[8] TypeScript [online]. Redmond, Washington, USA: Microsoft Corporation, 2022 [cit. 2022-05-02]. Dostupné z: https://www.typescriptlang.org/

[9] HTML: HyperText Markup Language. MDN [online]. Mountain View, CA, USA: Mozilla Foundation, 2022 [cit. 2022-05-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML

[10] What is CSS. MDN [online]. Mountain View, CA, USA: Mozilla Foundation, 2022 [cit. 2022-04-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS

[11] SQL - Overview. Tutorials Point [online]. Raleigh, NC, USA: Tutorials Point, 2022 [cit. 2022-04-16]. Dostupné z: https://www.tutorialspoint.com/sql/sql-overview.htm

[12] XML introduction. MDN [online]. Mountain View, CA, USA: Mozilla Foundation, 2022 [cit. 2022-04-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction

[13] Introducing JSON [online]. 2022 [cit. 2022-04-19]. Dostupné z: https://www.json.org/json-en.html

[14] A Deep Look at JSON vs. XML. Toptal [online]. Wilmington, DE, USA: Toptal, 2019 [cit. 2022-04-19]. Dostupné z: https://www.toptal.com/web/json-vs-xml-part-1

[15] What is a Framework?. Hackr.io [online]. Haryana, Indie: Hackr.io, 2022 [cit. 2022-04-17]. Dostupné z: https://hackr.io/blog/what-is-frameworks

[16] Framework. WhatIs.com [online]. Newton, MA, USA: WhatIs.com, 2022 [cit. 2022-04-17]. Dostupné z: https://www.techtarget.com/whatis/definition/framework

[17] What is .NET?. Microsoft .NET [online]. Redmond, Washington, USA: Microsoft Corporation, 2022 [cit. 2022-04-17]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet

[18] Introduction to .NET framework. Geeks for Geeks [online]. Noida, Uttar Pradesh, Indie: GeeksforGeeks, 2021 [cit. 2022-04-17]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-net-framework/

[19] What is .NET Framework?. Microsoft .NET [online]. Redmond, Washington, USA: Microsoft Corporation, 2022 [cit. 2022-05-02]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework

[20] Docker [online]. Palo Alto, CA, USA: Docker, 2022 [cit. 2022-05-02]. Dostupné z: https://www.docker.com/

[21] .NET Standard. Microsoft Documentation [online]. Redmond, Washington, USA: Microsoft Corporation, 2022 [cit. 2022-04-21]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/net-standard?tabs=net-standard-1-0

[22] Differences Between .NET Framework, .NET Core, and .NET Standard. CodeMaze [online]. Rawalpindi, Punjab, Indie: CodeMaze Pvt.Ltd, 2017 [cit. 2022-04-21]. Dostupné z: https://code-maze.com/differences-between-net-framework-net-core-and-net-standard/

[23] An introduction to NuGet. Microsoft documentation [online]. Redmond, Wa-shington, USA: Microsoft Corporation, 2022 [cit. 2022-04-17]. Dostupné z: https://docs.microsoft.com/en-us/nuget/what-is-nuget

[24] Language Integrated Query (LINQ). Microsoft documentation [online]. Red-mond, Washington, USA: Microsoft Corporation, 2022 [cit. 2022-04-17]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/linq/

[25] What is ASP.NET. Microsoft .NET [online]. Redmond, Washington, USA: Micro-soft Corporation, 2022 [cit. 2022-04-20]. Dostupné z: https://dotnet.micro-soft.com/en-us/learn/aspnet/what-is-aspnet

[26] Introduction to ASP.NET Identity. Microsoft Documentation [online]. Redmond, Washington, USA: Microsoft Corporation, 2022 [cit. 2022-04-26]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introdu-ction-to-aspnet-identity

[27] ASP.NET Core Blazor. Microsoft Documentation [online]. Redmond, Washing-ton, USA: Microsoft Corporation, 2022 [cit. 2022-04-20]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0

[28] Wiring. GitHub [online]. San Francisco, CA, USA: GitHub, 2022 [cit. 2022-04-16]. Dostupné z: https://github.com/WiringProject/Wiring

[29] About Wiring. Wiring [online]. Bogotá, Kolumbie: University of Los Andes, 2003 [cit. 2022-04-16]. Dostupné z: http://wiring.org.co/about.html

[30] What is Entity Framework?. Entity Framework Tutorial [online]. Indie: entityfra-meworktutorial.net, 2020 [cit. 2022-04-21]. Dostupné z: https://www.entityframe-worktutorial.net/what-is-entityframework.aspx

[31] Bootstrap [online]. San Francisco, California, USA: Bootstrap, 2022 [cit. 2022-05-03]. Dostupné z: https://getbootstrap.com/

[32] TailwindCSS [online]. Oklahoma City, OK, USA: Tailwind, 2022 [cit. 2022-05-03]. Dostupné z: https://tailwindcss.com/

[33] TailwindPlay [online]. Oklahoma City, OK, USA: Tailwind, 2022 [cit. 2022-05-03]. Dostupné z: https://play.tailwindcss.com/

[34] Introduction to Visual Studio. Geeks for Geeks [online]. Noida, Uttar Pradesh, USA: GeeksforGeeks, 2019 [cit. 2022-04-22]. Dostupné z: https://www.geeksforge-eks.org/introduction-to-visual-studio/

[35] Visual Studio Code [online]. Redmond, Washington, USA: Microsoft Corporation, 2022 [cit. 2022-04-22]. Dostupné z: https://code.visualstudio.com/

[36] Arduino IDE. JavaTpoint [online]. Noida, UP, Indie: JavaTpoint, 2021 [cit. 2022-04-22]. Dostupné z: https://www.javatpoint.com/arduino-ide

[37] What Is a Database?. Oracle Cloud Infrastructure [online]. Austin, Texas, USA: Oracle Corporation, 2022 [cit. 2022-04-22]. Dostupné z: https://www.oracle.com/database/what-is-database/

[38] The Types of Databases (with Examples). Matillion [online]. Manchester, UK: Matillion, 2020 [cit. 2022-04-22]. Dostupné z: https://www.matillion.com/resources/blog/the-types-of-databases-with-examples

[39] Microcontroller (MCU). TechTarget [online]. Newton, MA, USA: IoT Agenda, 2019 [cit. 2022-04-19]. Dostupné z: https://www.techtarget.com/iotagenda/definition/microcontroller

[40] What is Arduino?. Arduino [online]. Somerville, MA, USA: Arduino corporation, 2022 [cit. 2022-04-22]. Dostupné z: https://www.arduino.cc/en/Guide/Introduction

[41] Arduino Uno Rev3. In: Arduino store [online]. Somerville (HQ), MA, USAArduino, 2022 [cit. 2022-05-01]. Dostupné z: https://cdn.shopify.com/s/files/1/0438/4735/2471/products/A000066_01.iso_1000x750.jpg?v=1629815860

[42] ESP32 vs ESP8266 – Pros and Cons. Maker Advisor [online]. Gondomar, Portugalsko: RANDOM NERD TUTORIALS, UNIPESSOAL LDA, 2022 [cit. 2022-04-22]. Dostupné z: https://makeradvisor.com/esp32-vs-esp8266/

[43] ESP32. In: LáskaKit [online]. Rychnov nad Kněžnou, Česká Republika: Tereza Lásková, 2022 [cit. 2022-05-01]. Dostupné z: https://cdn.myshoptet.com/usr/www.laskakit.cz/user/shop/big/1202_foto-1.jpg?6137b46c

[44] Raspberry Pi [online]. Cambridge, UK: The Raspberry Pi Foundation, 2022 [cit. 2022-05-03]. Dostupné z: https://www.raspberrypi.org/

[45] What is a Raspberry Pi?. Opensource.com [online]. Raleigh, NC, USA: Red Hat, 2021 [cit. 2022-04-22]. Dostupné z: https://opensource.com/resources/raspberry-pi

[46] What is IoT?. Oracle [online]. Austin, Texas, USA: Oracle Corporation, 2022 [cit. 2022-04-22]. Dostupné z: https://www.oracle.com/cz/internet-of-things/what-is-iot/

[47] HTTP. MDN [online]. Mountain View, CA, USA: Mozilla Foundation, 2022 [cit. 2022-04-22]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP

[48] SSL. SSLS [online]. Praha, Česká republika: Alpiro, 2022 [cit. 2022-04-23]. Dostupné z: https://www.ssls.cz/slovnik/ssl.html

[49] TLS. SSLS [online]. Praha, Česká republika: Alpiro, 2022 [cit. 2022-04-23]. Dostupné z: https://www.ssls.cz/slovnik/tls.html

[50] WebAssembly Concepts. MDN [online]. Mountain View, CA, USA: Mozilla Foundation, 2022 [cit. 2022-04-23]. Dostupné z: https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts

[51] What Is Wi-Fi?. Cisco [online]. San Jose, CA, USA: Cisco Systems, 2022 [cit. 2022-04-23]. Dostupné z: https://www.cisco.com/c/en/us/products/wireless/what-is-wifi.html

[52] What Is Bluetooth? The Ultimate Guide. Lifewire [online]. New York, NY, USA: Lifewire, 2022 [cit. 2022-04-23]. Dostupné z: https://www.lifewire.com/what-is-bluetooth-2377412

[53] What is ZigBee?. Digi [online]. Hopkins, MN, USA: Digi International, 2022 [cit. 2022-04-23]. Dostupné z: https://www.digi.com/solutions/by-technology/zigbee-wireless-standard

[54] Bharath. The Clean Architecture—Beginner's Guide. Better Programming [online]. London, United Kingdom: Medium Corporation, 2022 [cit. 2022-04-23]. Dostupné z: https://betterprogramming.pub/the-clean-architecture-beginners-guide-e4b7058c1165

[55] FROST, Brad. Atomic design. Brad Frost [online]. Pittsburgh, PA, USA: Brad Frost, 2013 [cit. 2022-04-23]. Dostupné z: https://bradfrost.com/blog/post/atomic-web-design/

[56] Dependency Injection. TutorialsTeacher [online]. Indie: tutorialsteacher.com, 2022 [cit. 2022-04-23]. Dostupné z: https://www.tutorialsteacher.com/ioc/dependency-injection

[57] ČÁPKA, David. MVC architektura. ITNetwork [online]. Praha, Česká republika: David Čápka, 2016 [cit. 2022-04-23]. Dostupné z: https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor

[58] What is a REST API?. RedHat [online]. Raleigh, NC, USA: RedHat, 2020 [cit. 2022-04-23]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api

[59] What is an API?. MuleSoft [online]. San Francisco, CA, USA: MuleSoft, 2022 [cit. 2022-05-04]. Dostupné z: https://www.mulesoft.com/resources/api/what-is-an-api

[60] REST (REpresentational State Transfer). WhatIs.com [online]. Newton, MA, USA: WhatIs.com, 2022 [cit. 2022-04-23]. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/REST-REpresentational-State-Transfer

[61] Britannica, T. Editors of Encyclopaedia. "meteorology." Encyclopedia Britannica [online]. January 16, 2020 [cit. 2022-04-23]. Dostupné z: https://www.britannica.com/science/meteorology.

[62] Encyclopedia of Atmospheric Sciences. 1. Cambridge, Massachusetts, USA: Academic Press, 2003, s. 498-504. ISBN ISBN 9780122270901.

[63] OSWALD, Ed. What Is a Weather Station?. Weather Station Advisor [online]. Millersville, Pennsylvania, USA: Amazon Services, 2021 [cit. 2022-04-23]. Dostupné z: https://www.weatherstationadvisor.com/what-is-a-weather-station/

[64] WeeWX [online]. Tom Keffel, 2022 [cit. 2022-04-25]. Dostupné z: https://weewx.com/

[65] OpenWeather [online]. London, UK: OpenWeather, 2022 [cit. 2022-04-25]. Dostupné z: https://openweathermap.org/

[66] ThingSpeak [online]. Natick, Massachusetts, USA: MathWorks, 2022 [cit. 2022-04-25]. Dostupné z: https://thingspeak.com/

[67] Solar Powered WiFi Weather Station V3.0. In: Instructables [online]. San Rafael, Kalifornie, USA: Autodesk, 2022 [cit. 2022-04-25]. Dostupné z: https://www.instructables.com/Solar-Powered-WiFi-Weather-Station-V30/

[68] Open Weather Station. In: GitHub [online]. San Francisco, Kalifornie, USA: GitHub, 2022 [cit. 2022-04-25]. Dostupné z: https://github.com/panchazo/open-weather-station

[69] Use Cases. Usability.gov [online]. Washington, D.C., USA: U.S. General Services Administration, 2022 [cit. 2022-04-26]. Dostupné z: https://www.usability.gov/how-to-and-tools/methods/use-cases.html

[70] Kestrel web server. Microsoft .NET [online]. Redmond, Washington, USA: Microsoft Corporation, 2022 [cit. 2022-05-05]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-6.0

[71] Arduino Nano 33 IoT. In: Arduino store [online]. Somerville (HQ), MA, USA: Arduino, 2022 [cit. 2022-05-01]. Dostupné z: https://cdn.shopify.com/s/files/1/0438/4735/2471/products/ABX00027_01.iso_4b9706b9-00e7-4bad-a28e-2fe7244ac329_1000x750.jpg?v=1629821421

[72] ASP.NET Core Middleware. Microsoft Documentation [online]. Redmond, Washington, USA: Microsoft Corporation, 2022 [cit. 2022-05-01]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-6.0

[73] Swagger [online]. Somerville, MA, USA: SmartBear Software, 2021 [cit. 2022-05-06]. Dostupné z: https://swagger.io/

[74] MariaDB [online]. Middletown, Delaware, USA: MariaDB Foundation, 2009 [cit. 2022-05-06]. Dostupné z: https://mariadb.org/

[75] Duck DNS [online]. Amazon.com, Inc.: Seattle, Washington, USA, 2022 [cit. 2022-05-06]. Dostupné z: https://www.duckdns.org/

[76] ZeroSSL [online]. Vídeň, Rakousko: Stack Holdings, 2022 [cit. 2022-05-06]. Dostupné z: https://zerossl.com/

[77] OpenSSL [online]. Adamstown, Maryland, USA: OpenSSL Project, 1999 [cit. 2022-05-06]. Dostupné z: https://www.openssl.org/

[78] JWT.io [online]. Bellevue, WA, USA: Auth0, 2022 [cit. 2022-05-06]. Dostupné z: https://jwt.io/

[79] HMACSHA512 Class. Microsoft Documentation [online]. Redmond, Washington, USA: Microsoft Corporation, 2022 [cit. 2022-05-08]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.hmacsha512?view=net-6.0

[80] Security Considerations (Entity Framework). Microsoft Documentation [online]. Redmond, Washington, USA: Microsoft Corporation, 2022 [cit. 2022-05-08]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/security-considerations?redirectedfrom=MSDN

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

IoT	Internet of Things – Intenet Věcí
HTML	Hypertext Markup Language
CSS	Cascade Style Sheet
SQL	Structured Query Language
UI	User Interface
DIY	Do It Yourself
IDE	Integrated Development Environment
EF	Entity Framework
DBMS	Database Management System
USB	Universal Serial Bus
ARM	Advanced RISC Machines
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
MIMO	Multiple Input Multiple Output
IoC	Inversion of Control
SSL	Secure Socket Layer
TLS	Transport Layer Security
LINQ	Language-Integrated Query
JSON	JavaScript Object Notation
XML	Extensible Markup Language
CIL	Common Intermediate Language
CLR	Common Language Runtime
WPA	Wi-Fi Protected Access
WLAN	Wireless Local Area Network

DI	Dependency Injection
JWT	JSON Web Token
GUID	Globally Unique Identifier
API	Application Programming Interface
SKD	Software Development Kit
DNS	Domain Name Server
REST	Representational State Transfer

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD

Obsah přiloženého CD:

- Diplomová práce v elektronické podobě
- Zdrojové soubory vytvořeného systému
- Data z „Proof of concept“ systému
- Data ze současného systému