Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

PLZEŇ, 2020 Jakub Zieba

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne ....

Poděkování

Chtěl bych tímto poděkovat panu Ing. Janu Švecovi, Ph.D. za jeho ochotu a pečlivost při vedení této práce.

Jakub Zieba

Abstrakt

Tato práce se zabývá návrhem a vytvořením jednoduché hlasově ovládané meteostanice, jejíž součástí je hlasový dialogový systém využívající platformu *SpeechCloud*. Tento systém slouží k zodpovídání otázek týkajících se lokální předpovědi počasí získávané ze služby *OpenWeatherMap* a aktuálních a zaznamenaných hodnot teploty a vlhkosti získávaných pomocí připojeného senzoru.

Klíčová slova

meteostanice, hlasový dialogový systém, internet věcí, SpeechCloud, REST API, OpenWeatherMap, Raspberry Pi

Abstract

This paper describes the process of designing and creating a simple voice controlled weather station including a spoken dialog system which is using the *SpeechCloud* platform. This system's purpose is answering questions about local weather forecast coming from the *OpenWeatherMap* service and about current and past values of temperature and humidity acquired using the attached sensor.

Key words

weather station, spoken dialog system, internet of things, SpeechCloud, REST API, OpenWeatherMap, Raspberry Pi

Obsah

1 Úvod

Hlasové dialogové systémy se čím dál více stávají součástí našeho každodenního života. Téměř každý dnešní chytrý telefon je schopen vést omezenou konverzaci s jeho uživatelem. Systémy v telefonech nejen umožňují uživateli pomocí hlasových příkazů ovládat software v telefonu a nastavovat budík či přidávat události do kalendáře, ale také vyhledávat na internetu odpovědi na široké množství dotazů.

Využití hlasových dialogových systému však není omezené jen na mobilní telefony. Jsou dobře využitelné v celé řadě různých systémů od chytré domácnosti, přes asistenty pohybově či zrakově postižených osob, až po člověku podobné roboty.

Cílem této práce je na projektu jednoduché meteostanice demonstrovat využití hlasového dialogového systému pro centrální přístup k informacím získávaným z různých lokálních i internetových zdrojů. Zároveň tento projekt ukazuje dnešní možnosti distribuovaných systémů, které jsou založeny na principu distribuce jednotlivých úkolů na vícero libovolně vzdálených zařízení a komunikace mezi těmito zařízeními.

Aplikace běžící na Raspberry Pi, jejíž návrh je popsán v této práci, představuje centrální modul celého systému. Při stisknutí připojeného tlačítka zaznamená hlas uživatele a pošle jej k rozpoznání na externí službu *SpeechCloud*, která pošle zpět rozpoznaný dotaz. Aplikace tento dotaz zpracuje a vygeneruje odpověď, která se odešle znovu službě *SpeechCloud* pro syntézu řečového signálu, jenž je poté přehrán uživateli.

Informace, které aplikace umožňuje předávat uživateli, zahrnují přehledy záznamů teploty a vlhkosti ze senzoru, aktuální teplotu a vlhkost z téhož senzoru a předpověď počasí získávanou ze služby *OpenWeatherMap*.

2 Teoretická část

V této části je popsána teorie technologií použitých v projektu hlasově ovládané meteostanice a jeho jednotlivých součástí. Uvedená teorie se týká těchto čtyř oblastí:

- **Hlasové dialogové systémy, rozpoznávání/syntéza řeči**

  Nejprve je uvedena teorie za rozpoznáváním a syntézou řeči. Tyto oblasti tvoří důležitou součást hlasových dialogových systémů, které jsou popsány hned poté. Je zde zmíněna také platforma *SpeechCloud*, která je v projektu využívána pro rozpoznání, porozumění a syntézu řeči, čímž je zjednodušen návrh i implementace celého systému.

- **REST API**

  Dále je popsána architektura rozhraní REST. Toto rozhraní využívá projektem používaná služba *OpenWeatherMap.org*, která poskytuje údaje o předpovědi počasí. Byla zvolena pro svoje možnosti poskytování relativně kvalitních údajů zdarma a přes jednoduše použitelné REST API.

- **Databázové systémy, relační databáze**

  Pro trvalé uchování záznamů ze senzorů a předpovědi počasí slouží databáze, jelikož se jedná o nejlepší způsob vytvoření efektivně přístupného úložiště dat. Je zde popsán princip fungování databázových systémů, zejména těch relačních, a konkrétní v projektu použitá databáze MariaDB.

- **Mikropočítače Raspberry Pi**

  Celá aplikace i databáze běží na mikropočítači Raspberry Pi 2, proto je zde uvedena i kapitola popisující tato zařízení. Výhody použití právě tohoto zařízení jsou blíže specifikovány v praktické části této práce.

2.1 Rozpoznávání řeči

Rozpoznávání řeči označované zkratkou ASR (Automatic Speech Recognition) je proces převodu záznamu lidské řeči ve formě akustického signálu na text odpovídající zaznamenanému projevu řečníka.

2.1.1 Akustická analýza

Před samotným procesem rozpoznávání je potřeba převést akustický signál do digitální podoby. Nejjednodušší způsob tohoto převodu je *pulsní kódová modulace*:

Nejprve se signál navzorkuje předem zvolenou periodou. Důležité pravidlo pro volbu vzorkovací periody je Shannonův vzorkovací teorém, dle kterého musí být frekvence vzorkování minimálně 2x větší než nejvyšší frekvence obsažená v signálu, aby spektrum výsledného signálu odpovídalo původnímu. Po navzorkování se provede kvantizace a následné kódování. Jedná se o proces přiřazení hodnoty každého vzorku k nejbližší úrovni kvantování a má za následek omezení počtu možných hodnot, kterých můžou vzorky signálu nabývat. [1]

Často se dále provádí zpracování signálu v časo-frekvenční oblasti do formy tzv. *spektrogramu*. Jedná se o graf vyjadřující amplitudu jednotlivých frekvencí obsažených v jednotlivých časových úsecích signálu. K vypočtení hodnot spektrogramu se využívá krátkodobá Furierova transformace. Pomocí spektrogramu lze mnohem lépe rozpoznat jednotlivé hlásky než z pouhého časovém průběhu signálu.

Pro zpracování signálu se používají i další metody, mezi něž patří například *kepstrální analýza* sloužící k oddělení jednotlivých složek hlasových kmitů či *lineární prediktivní kódování*, které provádí odhad parametrů modelu řečové produkce.

2.1.2 Přístupy k rozpoznávání řeči

Volba způsobu rozpoznávání řeči ze zpracovaného akustického signálu je závislá na její konkrétní aplikaci. Pro hlasové dialogy s malým počtem příkazů se volí jiné metody rozpoznávání než pro systémy, jejichž účel je porozumět souvislé řeči. [2]

- **Rozpoznávání izolovaných slov**

  Tento přístup je jednodušší, avšak použitelný pouze pro malé množiny rozpoznávaných slov, tedy omezený slovník s maximálně několika desítkami slov. Používají se metody *srovnávání se vzorem* (template matching), které fungují na principu porovnávání záznamu řeči s uloženými referenčními vzory a zvolením toho nejpodobnějšího. Omezení na malou velikost slovníku je dáno potřebou existence jednoho či více vzorů pro každé slovo/promluvu a exponenciálně rostoucí časovou náročností algoritmu s přibývajícím počtem vzorů.

  Problémem srovnávání se vzorem bývá rozdílné časování (délka hlásek) v záznamu řeči a vzoru. Proto se pro porovnání používá algoritmus *dynamického borcení časové osy* (DTW = Dynamic Time Warping), který postupně mapuje příznaky rozpoznávaného slova na příznaky vzoru na základě jejich podobnosti. Umožňuje mapovat více příznaků rozpoznávaného slova na jedem příznak vzoru a opačně, čímž sjednotí časování.

- **Statistický přístup**

  Pro rozpoznávání souvislé řeči se využívá statistického přístupu, který je založen na hledání posloupnosti slov $\widehat{W}$, jež s nejvyšší pravděpodobností odpovídá rozpoznávané řeči O. Dle Bayesova vztahu se problém nalezení $P(W|O)$, tedy pravděpodobnosti, že nějaká posloupnost slov odpovídá určité rozpoznávané řeči, dá rozložit na problém nalezení $P(W)$, tedy pravděpodobnosti vyslovení dané posloupnosti slov řečníkem, a $P(O|W)$, neboli pravděpodobnosti, že z určité posloupnosti slov vznikne rozpoznávaná řeč:

$$\widehat{W} = arg \max_{W} P(W|O) = arg \max_{W} P(O|W) \cdot P(W)$$

  K nalezení $P(O|W)$ se používá *akustický model*, což je model vytváření řeči lidským hlasovými traktem. Tento model se skládá z modelů jednotlivých hlásek v kontextu hlásek okolních. Hlásky jsou modelovány pravděpodobnostmi výskytu a pořadí určitých zvuků, které se určují na základě trénovacích dat.

  Pro zjištění $P(W)$ slouží *jazykový model*, který určuje pravděpodobnosti výskytu slov v závislosti na slovech předchozích. Odhad těchto pravděpodobností se určuje na základě relativní četnosti jednotlivých posloupností slov v trénovacích datech. Vztah pro $P(W)$ v posloupnosti o K slovech je tedy:

$$P(W) = P(w_k|w_1, w_2, ..., w_{K-1}) = \prod_{i=1}^{K} P(w_i|w_1^{i-1})$$

  Jelikož by určování pravděpodobnosti výskytu slov na úplně všech předchozích slovech způsobilo příliš velkou komplexnost modelu, používá se tzv. *n-gramový model*, ve kterém se určuje pravděpodobnost výskytu pouze na předchozích $n-1$ slovech:

$$P(W) \approx \prod_{i=1}^{K} P(w_i|w_{i-n+1}^{i-1})$$

  Samotné nalezení nejpravděpodobnější posloupnosti slov $\widehat{W}$ poté odpovídá nalezení nejpravděpodobnější cesty grafem tvořeným kombinací akustického a jazykového modelu.

2.2 Syntéza řeči

Syntéza řeči je proces umělého vytváření řečového signálu z textu. Cílem je umožnit softwaru "přečíst"libovolný text způsobem co nejpodobnějším řeči člověka. Systémy umožňující převod textu na řeč se označují zkratkou TTS (Text To Speech). [3]

2.2.1 Analýza a zpracování textu

Před samotným procesem syntézy řeči je potřeba provést analýzu textu a zpracovat jej na posloupnost hlásek (fonémů). Tento proces se sestává z následujících fází:

1. **Předzpracování**

   Ve fázi předzpracování se detekuje typ a struktura textu. Odstraní se formátovací a bílé znaky. Dojde také k detekci jednotlivých slov, vět a odstavců.

2. **Normalizace**

   V této fázi dojde k přepisu čísel, zkratek, akronymů, symbolů, atd. do jejich plné slovní formy.

3. **Interpretace značkovaného textu**

   Zde se do textu přidají značky popisující určité vlastnosti syntetizované řeči. Jedná se například o styl čtení, tedy emoce (např. radost, smutek) či expresivní prvky (např. povzdechnutí).

4. **Lingvistická analýza**

   Lingvistická analýza slouží k jazykovému rozboru textu. Zahrnuje *analýzu morfologie*, tedy kompozici jednotlivých slov (nalezení předpony, kořene, základního tvaru slova, atd.). Pro zpřesnění morfologické analýzy se využívá *syntaktická analýza*, která z jednotlivých vět vytváří hierarchické struktury a určuje význam slov v kontextu věty.

   Dále lingvistická analýza obsahuje *POS tagování*, tj. určení mluvnických kategorií slov (pády, časy, rody, atd.), a *frázování*, což je rozdělení vět na jednotlivé fráze na základě interpunkce, funkčních a významových slov či statistických metod. Správné frázování zvyšuje přirozenost řeči.

5. **Fonetická transkripce**

   Jedná se o transkripci textu do fonetické podoby, tj. vytvoření posloupnosti fonémů z posloupnosti písmen. Při transkripci se využívají *fonetická pravidla* a *fonetický slovník*. Jejich využití závisí na jazyce syntetizované řeči. Například pro angličtinu je třeba fonetický slovník použít pro většinu slov, ale pro češtinu pouze pro výjimečné slovesnosti.

6. **Prozodická charakteristika**

   K posloupnosti fonémů se přiřadí prozodické značky popisující intonaci, časování, frázování a intenzitu. *Intonace* popisuje změny frekvence základního hlasivkového tónu, tj. výšky hlasu. *Časování* popisuje dobu trvání jednotlivých hlásek. *Frázování* určuje délku a pozici pauz v řeči a *intenzita* značí energii (hlasitost) jednotlivých hlásek.

2.2.2 Přístupy k syntéze řeči

Existuje mnoho přístupů k syntéze řeči ze zpracovaného textu. Tyto přístupy se rozdělují do dvou kategorií: *konkatenační syntéza*, která funguje na principu spojování tzv. *jednotek* řeči a *statistická parametrická syntéza*, jež generuje řeč z řečových modelů pomocí statistických metod či neuronových sítí.

- **Konkatenační syntéza** je signálový přístup, jenž provádí syntézu řeči pomocí konkatenace (řetězení) *řečových jednotek* uložených v inventáři. Tyto jednotky jsou částmi člověkem namluveného řečového signálu.

  Často používanou metodou konkatenační syntézy je *metoda výběrem jednotek* (unit selection), která vybírá nejvhodnější hlasovou jednotku z inventáře na základně fonetické a prozodické charakteristiky textu. Její úspěšnost silně závisí na velikosti a kvalitě inventáře. Řeč generována touto metodou působí často přirozeně, je však velice těžce modifikovatelná. Konkatenační syntéza je využívána ve většině komerčních systémů používajících syntézu řeči.

- **Statistická parametrická syntéza** je modelový přístup, jenž generuje řeč pomocí statistických modelů. V tomto přístupu se používá přirozená řeč pouze ke trénování statistického modelu, který poté ze zpracovaného textu generuje *řečové parametry*. Z těchto parametrů se dále syntetizuje řeč pomocí *vokodéru*. Dříve se pro modelování řeči používaly skryté Markovovy modely, dnes se však přechází na hluboké neuronové sítě.

  U této syntézy je na rozdíl od konkatenačního přístupu možné dosáhnout dobré kvality i s menším počtem dat. Další výhodou je větší modifikovatelnost řeči díky možnosti měnit parametry modelu. Řeč však není tolik kvalitní a nepůsobí tak přirozeně jako u metody výběru jednotek.

2.3 Hlasové dialogové systémy

Účelem hlasových dialogových systému je zprostředkování hlasového rozhraní mezi uživatelem a softwarem. Tyto systémy využívají rozpoznávání a syntézu řeči k vedení hlasového dialogu téměř vždy omezeného na určitou aplikační oblast. Zvolenou oblastí může být například objednávání jídla, správa kalendáře či předpověď počasí. [4]

2.3.1 Popis jednotlivých modulů

Funkci hlasového dialogového systému můžeme popsat cyklem tvořeným pěti propojenými moduly, na jehož počátku i konci je uživatel. [4] [5]

1. **Rozpoznávání řeči**

   Jako první dojde k rozpoznání promluvy uživatele a převedení do textové podoby (viz kapitola 2.1).

2. **Porozumění řeči**

   Dále je potřeba větě od uživatele porozumět. *Znalostní přístup* k porozumění řeči spočívá v definici bezkontextových gramatik, pomocí kterých se přiřazuje význam určitým předem definovaným sématickým entitám. *Statistický přístup* je založen na odhadu významu věty pomocí natrénovaného modelu. Výsledkem procesu porozumění je významová reprezentace řeči.

3. **Dialogový manažer**

   Funkcí dialogového manažeru je na základě pozorování *stavu uživatele* reprezentovaného významovým popisem řeči a vnitřním stavem dialogu provést aktualizaci stavu dialogu a vygenerovat odpověď ve významové reprezentaci.

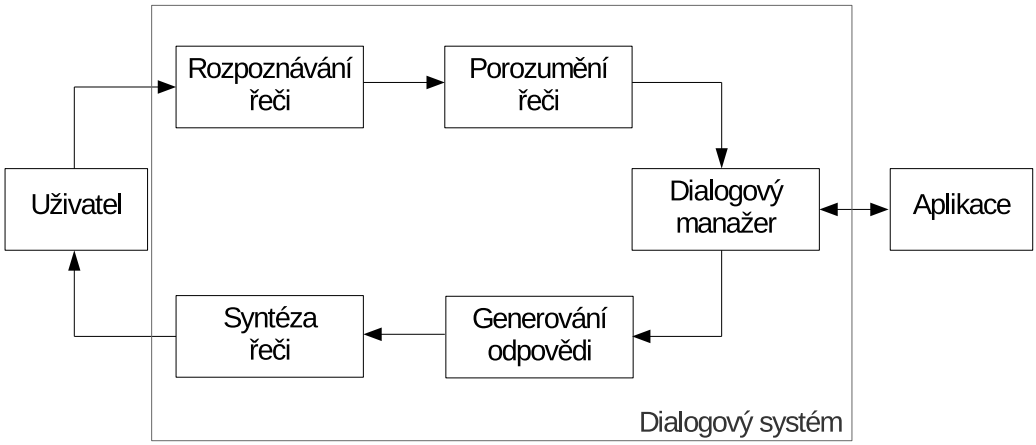
   Pro aktualizaci stavu dialogu a generaci odpovědi potřebuje manažer často provést další akce, zpravidla využitím dalších modulů systému, které nejsou přímo součástí obsluhy hlasového dialogu. Těmito akcemi může být například komunikace s API, databází, řídícím systémem, atd.

4. **Generování odpovědi**

   Významovou reprezentaci odpovědi je poté potřeba převést na text. Převod se často implementuje jako pouhé dosazování do předdefinovaných šablon. Alternativou jsou komplexnější metody pro generování přirozeného textu na základě jazykových modelů (NGL = Natural-Language Generation).

5. **Syntéza řeči**

   Nakonec dojde k syntéze hlasového signálu z textové odpovědi a jeho přehrání uživateli (viz kapitola 2.2).



Obrázek 1: Schéma hlasového dialogového systému

2.3.2 SpeechCloud

Platforma SpeechCloud umožňuje realizaci rozpoznávání, porozumění a syntézy řeči a je tedy vhodná jako základ pro hlasové dialogové systémy.

Pro každý dialogový systém je vytvořena nezávislá instance umožňující klientovi zasílat záznam řeči k rozpoznání a přijímat syntetizovanou řeč. Obojí probíhá pomocí protokolu SIP, který slouží pro zprostředkování obousměrné hlasové komunikace. Řídící komunikace, jež zahrnuje zasílání významové reprezentace rozpoznávaného textu či textu pro syntézu, probíhá přes oddělené rozhraní. [6]

SpeechCloud využívá pro porozumění řeči *gramatiky* zadávány v ABNF formátu. Tyto gramatiky představují soubor pravidel pro přiřazení významu určitým sématickým entitám (slovům). Příklad pravidla pro porozumění ročnímu období:

```
$rocniObdobi = (
    (jaro | jaře) {1} |
    (léto | létě) {2} |
    podzim {3} |
    (zima | zimě) {4}
);
```

Tento zápis definuje pravidlo jménem `rocniObdobi`. Oddělovač "|"slouží pro oddělení alternativ slov. Toto pravidlo bude tedy aplikováno na každé slovo odpovídající jedné z definovaných alternativ. Složené závorky představují tagy, které označují jednotlivé alternativy pro určení, jaká z nich byla použita při aplikaci pravidla. Zde jsou jako tagy použita čísla, je však možné použít i slova.[7]

2.4 REST API

REST API je softwarová architektura programového rozhraní (API = Application Programming Interface) navržena pro komunikaci v distribuovaném prostředí s aplikacemi poskytující určité zdroje.

Architekturu REST (REpresentational State Transfer) navrhl ve své disertační práci v roce 2000 američan Roy Thomas Fielding, [8] jenž je zároveň spoluzakladatelem projektu Apache HTTP Server, což je podle průzkumu W3Techs z dubna 2020 nejpoužívanější webový server na světě. [9]

2.4.1 Zdroje

Zdroj (*resource*) je abstraktní označení pro jakoukoliv informaci poskytovanou přes REST rozhraní. Podle definice z Fieldingovy disertační práce je zdroj R časově závislou funkcí $M_R(t)$, která v čase t poskytuje množinu obsahující buď konkrétní reprezentace zdrojů, identifikátory odkazující na zdroje, nebo kombinaci obojího. Může se také jednat o prázdnou množinu. Každý zdroj má vlastní unikátní identifikátor, pomocí kterého je k němu přistupováno. [8]

Zdroj závislý na čase je například "Aktuální počasí v Praze". Zdroje, jež jsou nezávislé na čase, například "Datum vydání knihy XY"jsou označovány jako *statické* a poskytují vždy stejnou množinu hodnot.

Stav zdroje v konkrétním čase se nazývá *reprezentace*. Obecně se jedná o libovolnou posloupnost bajtů. Reprezentace obsahuje *metadata* ve formátu klíč-hodnota, která popisují způsob interpretace této posloupnosti. Tedy zda se jedná o text, obrázek, kolekci reprezentací, atd. Metadata zpravidla obsahují i další informace o reprezentaci a jejím zdroji.

2.4.2 Principy architektury

Struktura rozhraní REST je postavena na následujících šesti principech. Rozhraní musí splňovat prvních 5 z těchto principů (poslední princip je volitelný), aby mohlo být označeno jako RESTful, což je označení pro rozhraní používající architekturu REST. [8]

- **Klient-Server:**

  Tento princip je převzat ze stejnojmenné architektury. Jedná se o rozdělení datového úložiště a uživatelského rozhraní na samostatné instance. To zjednodušuje převod uživatelského rozhraní na jiné platformy a umožňuje nezávislý vývoj obou instancí.

- **Bezestavovost:**

  Komunikace klient-server musí být dle tohoto principu bezestavová, tj. každý dotaz na server obsahuje veškeré potřebné informace a server nemůže používat informace o předchozích dotazech ke zpracování aktuálního. Klient však může použít informace z předchozích dotazů a odpovědí (pokud označeny jako *cacheable*) při posílání nových dotazů.

- **Caching:**

  Pokud je odpověď ze serveru označena jako *cacheable*, může si ji klient uschovat a data z ní použít pro pozdější účely. To má za následek snížení počtu potřebných dotazů na server, což vede ke zvýšení efektivity klienta i serveru. Problém může nastat, pokud se uložená data příliš liší od těch, která by byla získána při novém dotazu na server. Částečným řešením může být pravidelné promazávání starších uložených dat.

- **Jednotné rozhraní:**

  Pro jednoduchou tvorbu jednotlivých komponent a usnadnění komunikace mezi nimi bylo zavedeno jednotné rozhraní. Toto rozhraní je postaveno na několika konceptech: identifikace zdrojů, která umožňuje přístup ke konkrétnímu zdroji pomocí identifikátoru; možnost změnit stav určitého zdroje pomocí informací získaných z jeho reprezentace; jednoznačnost zpráv, tedy aby každá zpráva obsahovala dost informací k tomu, aby ji bylo možné správně přečíst; možnost použití odkazů na jiné zdroje v odpovědích na dotazy.

- **Rozdělení do vrstev:**

  Tento princip dovoluje rozdělení jednotlivých komponent do vrstev tak, že každá komponenta přistupuje vždy pouze k té vrstvě komponenty se kterou může interagovat a nemá přístup do "hlubších"vrstev. Tím je možné zvýšit bezpečnost komunikace a celkovou modularitu.

- **Kód na vyžádání:**

  Princip kódu na vyžádání umožňuje klientovi dynamicky stahovat a spouštět kód poskytnutý serverem. Tím dojde k značnému zjednodušení klienta a přidání možnosti implementovat nové funkce bez nutnosti vydání jeho nové verze.

2.4.3 Průběh komunikace

Komunikace s RESTful API většinou probíhá přes HTTP protokol. Toto však není nutnost, jelikož ve specifikaci REST není určen žádný konkrétní protokol. [8] Pokud budeme předpokládat použití HTTP protokolu, probíhá komunikace následujícím způsobem:

Klient pošle dotaz ve specifickém formátu na server. Dotaz zpravidla obsahuje identifikátor zdroje ve formátu URI (Uniform Resource Identifier) a je zaslán pomocí jedné z HTTP metod. Nejčastěji používané metody jsou: [10]

- **GET** - vyžádání reprezentace určitého zdroje. Tato metoda nemění stav zdroje na serveru.
- **PUT** - aktualizace stavu zdroje.
- **DELETE** - odstranění zdroje.
- **POST** - vytvoření nového zdroje či přidání nových dat do existujícího. Někdy se používá i k aktualizaci a odstranění zdroje místo metod PUT a DELETE.

Po přijetí dotazu serverem dojde k jeho zpracování a odešle se odpověď klientovi. Odpovědí je vždy stavový kód a případně i reprezentace zdroje.

Stavové kódy jsou tříciferná čísla, jejichž význam je definován HTTP protokolem, a jsou rozděleny do skupin dle první cifry. Kódy začínající číslicí 1 jsou pouze informativní a oznamují, že dotaz byl přijat. Číslicí 2 se označuje úspěšné přijetí, zpracování a provedení požadovaných akcí. Číslice 3 oznamuje klientovi, aby provedl nějakou činnost k úspěšnému zpracování dotazu. Například pokud se identifikátor zdroje změnil a musí být použit ten nový. Chyby se označují číslicí 4, pokud byly způsobeny klientem (např. špatný formát dotazu) a číslicí 5, pokud byly způsobeny serverem. [11]

2.4.4 OpenWeatherMap.org

OpenWeatherMap (OWM) je webová služba spravována společností OpenWeather Ltd., která poskytuje informace o počasí v minulosti, v současnosti a předpověď počasí v budoucnu. Tyto informace jsou poskytovány pomocí několika REST API, z nichž ne všechna jsou přístupná zdarma. [12]

Informace o počasí obsahují teplotu, atmosferický tlak, vlhkost, rychlost větru a typ počasí (zataženo, slunečno, déšť, atd.)

OWM získává data o předpovědi počasí z několika modelů, především se jedná o model NOAA GFS (Spojené státy) a modely jež poskytují Environment Canada a ECMWF (European Centre for Medium-Range Weather Forecasts). Toto nejsou však jediné modely, jež OWM používá. [13]

Ke zpracování dat využívá OWM platformu VANE, která je schopná efektivně zpracovávat velké množství dat o počasí. OWM ji využívá především ke kombinování jednotlivých lokálních a globálních modelů pro získání přesnějších modelů ve specifických oblastech. [13]

2.5 Databáze

2.5.1 Databázové systémy

Databáze je organizovaný soubor dat s pevnou strukturou uložený v elektronické podobě. Každá databáze zpravidla potřebuje *systém pro řízení báze dat* (DBMS = DataBase Management System), který tvoří rozhraní mezi bází dat a softwarem, jenž databázi využívá. Jeho hlavním úkolem je zprostředkovat přidávání, odebírání a aktualizaci dat v databázi za dodržení dané struktury. DBMS se společně s databází označuje jako *databázový systém*. [14]

2.5.2 Relační databáze

Existuje mnoho modelů pro strukturu databázového systému, nejpoužívanějším je stále tzv. *relační databáze*. Tento model navrhl angličan Edgar F. Codd v 70. letech a stal se oblíbeným v 80. letech.

V relační databázi jsou data uspořádána do *tabulek*, což jsou dvourozměrné struktury sestávající se z *řádek* a *sloupců*. Každá řádka představuje uspořádanou n-tici prvků odpovídajících jednotlivým atributům (sloupcům) tabulky. Na pořadí řádků nezáleží. Aby však bylo možné správně interpretovat jednotlivé prvky řádků, musí být zachováno pořadí sloupců. Každý sloupec má přiřazený *datový typ*, který určuje způsob interpretace jednotlivých prvků. Tedy, zda se jedná o celé číslo, desetinné číslo, text, atd.

Jelikož na pořadí řádků nezáleží, zavádí se v relační databázi pro identifikaci konkrétních řádků tzv. *primární klíč*, který přiřazuje unikátní hodnotu ke každému řádku v tabulce. Hodnotou primárního klíče je umožněno referovat na určitý řádek programům komunikujících s databází nebo jiným tabulkám v databázi. Primárním klíčem může být již existující sloupec (pokud jsou jeho hodnoty unikátní), nicméně se častěji zavádí nějaký arbitrální unikátní identifikátor, například osobní číslo studenta na univerzitě. [15]

2.5.3 SQL

SQL (Structured Query Language) je programovací jazyk vytvořený v roce 1970 pro práci s relačními databázemi a je stále nejpoužívanějším jazykem k tomuto účelu. Patří mezi *deklarativní jazyky*, což je označení pro programovací jazyky sloužící pouze pro definici akce, kterou chceme provést, a o samotném způsobu provedení rozhoduje konkrétní implementace. Uživatel pomocí SQL pouze určí, jaké změny chce v databázi provést, a neurčuje, jak se provedou.

SQL můžeme rozdělit na několik "podjazyků", z nichž každý zastává jinou funkci. Tyto "podjazyky"však nejsou oddělené a je možné je libovolně kombinovat. Některé příkazy je možné zařadit do více z nich zároveň. [16]

- **DDL - Data Definition Language** slouží k vytvoření/mazání databáze a definici její struktury. Pomocí DDL můžeme vytvářet, mazat a upravovat strukturu tabulek či jiných databázových objektů.

- **DML - Data Modification Language** slouží pro modifikaci dat (řádků v tabulkách) v databázi, tedy k jejich vkládání, úpravě a mazání.

- **DQL - Data Query Language** slouží pro čtení dat z databáze pomocí širokých možností popisu požadovaných dat (z jaké tabulky chceme číst, jakou podmínku musí data splňovat, atd.) a formátu ve kterém jsou předány (řazení, seskupení, atd.). [17] Někdy se označuje jako součást DML.

- **DCL - Data Control Language** slouží pro správu autorizace přístupu k databázi. Umožňuje povolit/zakázat danému uživateli úplný nebo omezený přístup k určitému zdroji. Částečný přístup může znamenat například povolení DML a DQL (možnost vkládat, editovat a číst záznamy), ale zákaz DDL (změna struktury databáze). DCL není součástí všech implementací SQL. Například databáze SQLite omezení přístupu pomocí DCL nepodporuje.

- **TCL - Transaction Control Language** slouží pro zvýšení bezpečnosti úprav v databázi tím, že umožňuje spojení většího množství příkazů do jedné tzv. *transakce*. Jednotlivé příkazy v transakci se provádějí pouze na kopii databáze a v originální databázi se projeví až po potvrzení změn (*commit*). Pokud některý z příkazů v transakci selže, kopie databáze se vrátí do původního stavu před transakcí.

  Transakce tedy zaručuje atomizaci změn v databázi. Pokud převedeme peníze z jednoho účtu na jiný, potřebujeme, aby se provedly buď obě operace (odebrání částky z jednoho účtu a přidání na jiný) nebo ani jedna z nich.

2.5.4 MariaDB

MariaDB je jednou z implementací relační SQL databáze vyvíjené od roku 2009 společností MariaDB Foundation. Je plně open-source a distribuována pod licencí GPL (GNU General Public Licence). [18] Z toho důvodu se jedná o výchozí databázi ve většině distribucí operačního systému Linux a je proto i často používána webovými aplikacemi. [19] MariaDB je větví stále používané a podporované databáze MySQL, která byla vytvořena, když databáze MySQL změnila licenci a přestala být plně open-source.

2.5.5 Jiné databázové modely

Relační model je nejrozšířenější, ne však jediný model databázového systému. Jeden z prvních modelů, který se dnes již téměř nepoužívá, je *hierarchický model*. Ten byl navržen v roce 1960 a je založen na jednoduché stromové struktuře a vztahu rodič-potomek. Každý rodič může mít více potomků, ale každý potomek jen jednoho rodiče. Tento model tedy podporuje pouze vztahy 1:1 a 1:N. Nejznámější dodnes používanou hierarchickou databází je registr operačního systému Windows. [20]

Hierarchický model byl později nahrazen *síťovým modelem* navrženým v roce 1969, jenž funguje na podobném principu, nicméně podporuje i vztahy N:N. Od podpory N:N vztahů bylo později znovu upuštěno v již zmíněném relačním modelu.

S rostoucí oblibou objektově orientovaného programování byl v 80. letech navržen *objektově orientovaný model*, který místo tabulek používá objekty a místo sloupců atributy těchto objektů. Výhoda objektově orientovaných databází je jejich integrace s programovacím jazykem aplikace spravující tuto databázi.

Populární je dnes také typ databází označovaný jako NoSQL. Jedná se o databáze, jejichž architektura neodpovídá žádnému databázovému modelu. Bývají navrženy speciálně pro konkrétní typ aplikace, čímž dosahují vyšší efektivity, především při práci s velkým množstvím dat (tzv. *big data*). [14]

2.6 Raspberry Pi

Raspberry Pi je série malých jednodeskových počítačů navržena a vyráběna britskou společností Raspberry Pi Foundation. Prvotní myšlenka byla vytvořit levné zařízení pro studenty, které jim pomůže lépe porozumět fungování počítačů. [21] Avšak díky nízké ceně si Raspberry Pi rychle získalo popularitu a proniklo do mnoha oblastí kybernetiky.

Prvními verzemi zařízení vytvořenými v roce 2012 byl Raspberry Pi Model B a o rok pozdější levnější varianta Model A. Tyto verze se již neprodávají, jelikož je v roce 2014 je nahradily zmenšené varianty Model B+ a Model A+.

V roce 2015 byla vytvořena verze Raspberry Pi 2 s výkonnějším procesorem a větší kapacitou RAM. Ve stejném roce se začala prodávat i verze Raspberry Pi Zero, která je mnohem menší než ostatní verze.

Pozdější verze, tedy Raspberry Pi 3, obsahuje nejen výkonnější procesor než předchozí verze, ale i vestavěnou WiFi a Bluetooth. Začala se vyrábět v roce 2016. Vestavěná WiFi a Bluetooth je přítomna i ve verzi Raspberry Pi Zero W vytvořené v roce 2018. Tento rok se začaly prodávat i varianty Raspberry Pi 3 Model B+ a menší Model A+.

Nejnovější verzí je nyní Raspberry Pi 4. Oproti předchozím verzím obsahuje navíc USB 3.0 porty, dva micro HDMI porty, které podporují rozlišení max 4K, a také napájení pomocí USB-C. [22]

2.6.1 Hardware

Všechny modely Raspberry Pi obsahují procesor s ARM architekturou od společnosti Broadcom, jehož specifikace se v jednotlivých modelech liší. Co se týče RAM, čím novější verze, tím větší kapacitu RAM má. První verze mají 512 MiB RAM, verze 2 a 3 mají 1 GiB RAM a nejnovější verze 4 je dostupná s 2, 4 nebo 8 GiB RAM.

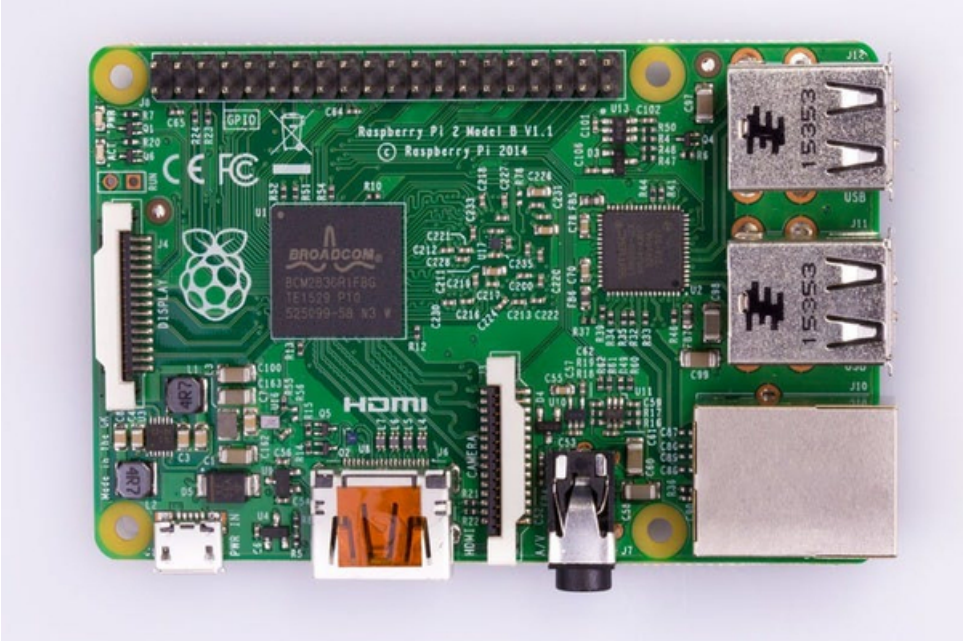
V jednotlivých modelech se liší i vstupní/výstupní hardware. Většina modelů obsahuje 4 USB porty, kromě modelů A a A+, které mají pouze 1, a modelů Zero, u nichž byly nahrazeny micro USB portem. Všechny verze obsahují HDMI nebo micro HDMI port připojen k vestavěnému grafickému procesoru od společnosti Broadcom.

Raspberry Pi umožňuje digitální zvukový výstup přes HDMI a analogový přes 3,5 mm konektor. Analogový výstup není dostupný v modelech Zero.

Veškeré modely kromě A, A+ a Zero mají síťový ethernetový port. Pouze Raspberry Pi 4 však podporuje plnou rychlost 1 Gbit/s. Verze 3 a novější obsahují navíc i čipy pro WiFi a Bluetooth (včetně verze Zero W).

Data jsou uložena na MicroSD kartě. Je možné použít libovolnou MicroSD kartu, avšak kapacita větší než 256 GiB je podporována až od verze 3 B+. [23] [22]

Důležitou součástí Raspberry Pi je GPIO (General Purpose Input-Output) konektor. Ten obsahuje 40 pinů, jejichž využití závisí z velké části na vůli programátora konkrétní aplikace. Tyto piny jsou využívány pro jednoduché připojení vlastního hardwaru (např. senzoru, displeje, atd.). GPIO obsahuje piny pro napájení (3,3V a 5V), přenos dat (vstup nebo výstup dle konfigurace) a uzemnění. Některé datové piny podporují PWM (pulzně-šířková modulace) a určité piny mohou být využity pro konkrétní sériové sběrnice (SPI a I2C). [24]



Obrázek 2: Raspberry Pi 2

2.6.2 Software

Součástí Raspberry Pi je operační systém Raspberry Pi OS dříve nesoucí název Raspbian. Je založen na linuxové distribuci Debian a optimalizovaný pro zařízení Raspberry Pi. Již zkompilované aplikace pro Raspberry Pi OS jsou dostupné k instalaci z oficiálního repozitáře pomocí balíčkovacího systému APT, jenž se používá i v Debianu. [25]

Raspberry Pi OS propaguje psaní softwaru v jazyce Python a je to díky tomu často používaný programovací jazyk na Raspberry Pi. Operační systém obsahuje i výchozí podporu různých programovacích nástrojů, jako je například Scratch, který slouží pro jednoduchou tvorbu animací a her. [26]

Na Raspberry Pi je však možné nainstalovat i jiný operační systém, za podmínky, že podporuje jeho hardware.

3 Praktická část

Zadáním této práce bylo navrhnout a realizovat integraci vhodných environmentálních senzorů s programovatelnou vývojovou deskou a strukturu hlasového dialogového systému využívajícího platformu *SpeechCloud* a poskytujícího údaje z těchto senzorů. Pro realizaci tohoto zadání byla navržena aplikace obsahující tyto funkcionality:

- Čtení dat o teplotě a vlhkosti ze senzoru
- Získávání předpovědi počasí z internetu přes API
- Ukládání údajů ze senzoru a z API do databáze
- Rozpoznání a porozumění řečovým dotazům uživatele
- Generování a syntéza odpovědi na dotaz na základě uložených dat
- Logování oznámení a chybových hlášek do souborů

3.1 Volba a zapojení hardware

Pro realizaci aplikace s výše zmíněnými funkcionalitami byl zvolen následující hardware:

- **Raspberry Pi 2**

  Jako centrální komponent byla zvolena programovatelná vývojová deska Raspberry Pi 2. Výhodou tohoto jednočipového mikropočítače je použití distribuce operačního systému Linux, který podporuje velké množství softwaru a zvyšuje tak možnosti využití. Důležitá je pro tento projekt i existence GPIO pinů především pro připojení senzoru. Raspberry Pi 2 obsahuje dostatečně výkonný hardware pro běh nejen celé aplikace, ale i databáze MariaDB. Na Raspberry Pi 2 je dále referováno jako na "zařízení".

- **Wi-Fi adaptér**

  Jelikož Raspberry Pi 2 neobsahuje integrovaný Wi-Fi modul, je nutné pro Wi-Fi připojení použít externí adaptér připojený k zařízení pomocí USB portu.

- **Mikrofon a reproduktor**

  Pro zvukový vstup a výstup je třeba mít k zařízení připojený mikrofon a reproduktor pomocí USB portu. Při vývoji aplikace byla používána sluchátka s mikrofonem, avšak pro praktické použití projektu by byl vhodnější samostatný mikrofon a reproduktor.

- **Senzor teploty a vlhkosti DHT11**

  Pro měření environmentálních veličin byl zvolen senzor teploty a vlhkosti DHT11. Výhodou tohoto senzoru je jeho jednoduché zapojení díky možnosti posílat data, která obsahují hodnoty dvou různých veličin, pouze po jednom vodiči pomocí protokolu 1-Wire.

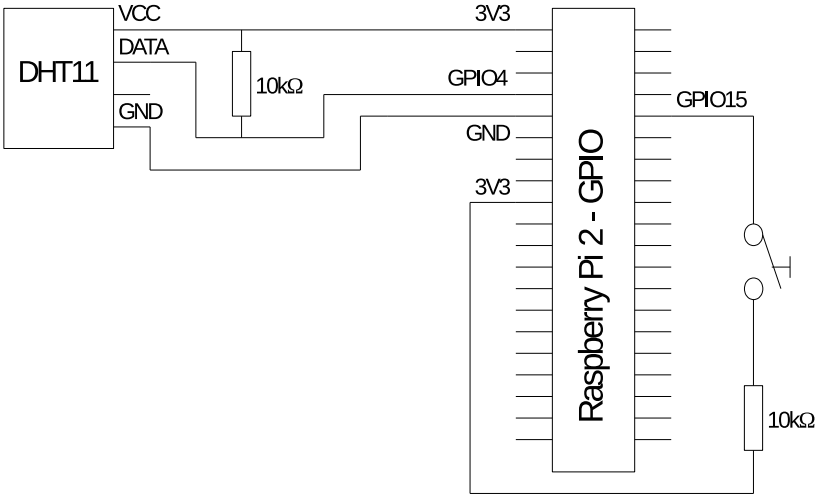
  Tento senzor umožňuje měření teploty v rozmezí 0 - 50 °C s přesností ±1°C a vlhkost v rozmezí 20 - 90% s přesností ±4%. Tato rozmezí jsou zcela dostačující například pro měření teploty a vlhkosti v místnosti. Pro jiné aplikace, například měření venkovní teploty, by bylo potřeba použít jiný senzor, který je schopný měřit i záporné hodnoty teploty (za předpokladu, že v tom místě dosahují přes zimu záporných hodnot).

  Senzor je připojen přímo na piny GPIO konektoru zařízení pomocí třech vodičů. Jeden slouží pro napájení senzoru napětím 3,3V (`VCC` → `3V3`), druhý pro uzemnění (`GND` → `GND`) a třetí pro přenos dat pomocí 1-Wire protokolu (`DATA`→ `GPIO4`). Mezi vodiči pro napájení a přenos dat je vložen pull-up rezistor 10kΩ, který slouží ke snížení šumu tím, že "táhne"napětí na datovém vodiči nahoru k hodnotě HIGH.

- **Tlačítko pro aktivaci rozpoznávání hlasu**

  Aby nedocházelo ke zbytečným pokusům o rozpoznávání dotazů při běhu aplikace, bylo zvoleno použití tlačítka pro aktivaci záznamu hlasu a následné rozpoznání. Při stisknutí tlačítka se ozve zpráva "Prosím, mluvte"a po odeznění této zprávy je možné zadat dotaz.

  Tlačítko je připojeno mezi piny `3V3` a `GPIO15`. Stisknutí tlačítka je detekováno čtením hodnoty napětí na pinu `GPIO15` (HIGH = je stisknuto, LOW = není stisknuto).



Obrázek 3: Schéma zapojení senzoru a tlačítka

3.2 Návrh software

Na zařízení se automaticky při jeho zapnutí spustí aplikace napsána v programovacím jazyce Python 3. Tento jazyk byl zvolen především z důvodu, že pro něj existuje knihovna pro komunikaci s platformou *SpeechCloud*. Jeho výhodou je také existence jednoduše použitelných knihoven pro komunikaci s REST API, databází a periferiemi Raspberry Pi.

Aplikace při zapnutí spustí dvě paralelní vlákna (thread), z nichž první slouží pro pravidelné získávání a ukládání dat a druhé pro vedení hlasového dialogu s uživatelem. Toto rozdělení je důležité z důvodu, že komunikace s externím hardwarem či službami není instantní. Pokud by celý program běžel v jednom vlákně, muselo by se se získáváním dat pokaždé vyčkat, než *SpeechCloud* dokončí rozpoznávání/syntézu řeči, a naopak komunikace s platformou *SpeechCloud* by musela čekat na přijetí dat ze senzoru či API.

3.2.1 Systém pro získávání dat

Pro pravidelné získávání dat slouží programová smyčka, která s periodou 10 minut získává údaj o teplotě a vlhkosti ze senzoru DHT11 a s periodou 2 hodiny předpověď počasí ze služby *OpenWeatherMap*. Hlídání periody je realizováno detekcí násobků periody v systémovém čase. Kvůli nekonstantnímu malému zpoždění při čtení ze senzoru není perioda 10 minut dodržena vždy přesně. Toto je řešeno uložením záznamu společně s datem a časem jeho přijetí. U předpovědi počasí není zcela přesné dodržení periody důležité. Záznamy z obou zdrojů jsou hned po přijetí uloženy do databáze MariaDB.

3.2.2 Hlasový dialogový systém

Vedení hlasového dialogu s uživatelem probíhá následujícím způsobem: Po stisknutí tlačítka dojde k záznamu řeči uživatele a jejímu následnému rozpoznání a porozumění platformou *SpeechCloud*, tedy převedení na dotaz ve významové reprezentaci.

Tento dotaz se následně zpracuje a vytvoří se odpověď. Dosazením do šablon se poté vygeneruje odpověď ve formě věty a *SpeechCloud* provede syntézu na řečový signál, který se přehraje uživateli.

Hlasový dialog je bezestavový, tedy že žádná otázka od uživatele nemá vliv na způsob zpracování otázek následujících a uživatel může položit libovolnou podporovanou otázku v libovolném stádiu dialogu. Jelikož je účelem dialogu pouze zodpovídání na sobě nezávislých otázek, nebylo použití komplexnější struktury s více stavy nutné.

Dialog podporuje 3 druhy řečových dotazů:

- **Předpověď počasí**

  Dotazy na předpověď počasí. Je možné se ptát na den relativně k dnešnímu (zítra, pozítří,...) nebo názvem dne v týdnu (pondělí, úterý,...). Lze se také zeptat na konkrétní počasí (déšť, sněžení,...).

  Jaké bude [den] *počasí*? (př: *Jaké* bude pozítří počasí?)

  Jaké bude [den] [denní doba] *počasí*? (př: *Jak* bude zítra odpoledne?)

  Bude [den] [denní doba] [počasí]? (př: Bude v neděli ráno pršet?)

  Jaká bude [den] [denní doba] teplota? (př: *Jak* bude v úterý ráno teplo?)

- **Záznamy ze senzoru**

  Dotazy na záznamy teploty a vlhkosti. Stejně jako u předpovědi je možné se ptát na relativní den (včera, předevčírem,...) nebo názvem dne v týdnu (pondělí, úterý, ...).

  Jaká byla [den] [veličina]? (př: *Jaká* byla v pondělí teplota?)

  Jaká byla [den] [denní doba] [veličina]? (př: *Jak* bylo včera ráno vlhko?)

- **Aktuální údaje ze senzoru**

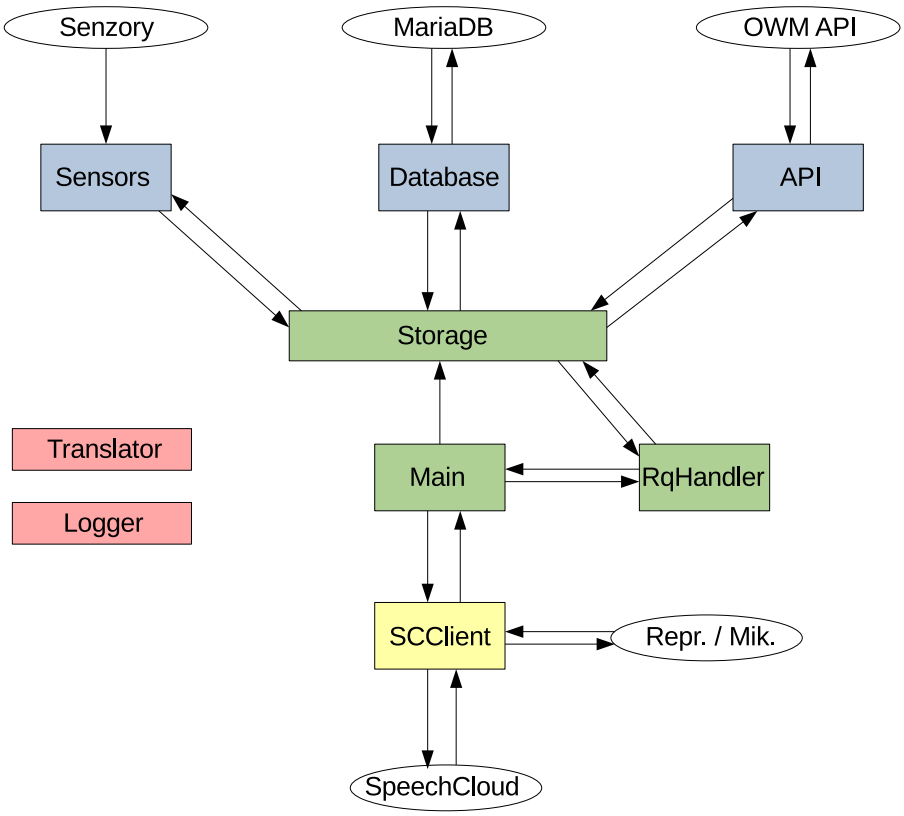
  Dotazy na aktuální teplotu a vlhkost.

  Jaká je *venku* [veličina]? (př: *Jaká* je *venku* teplota?)

Slova označena *kurzívou* nejsou důležitá pro porozumění a jsou zde uvedena pouze pro přehlednost a přirozenost otázky. Proces porozumění je podrobněji popsán v kapitole 3.5 a zpracování i generace odpovědi v kapitole 3.7.

3.3 Struktura aplikace

Aplikace je rozdělena do jednotlivých modulů, které spolu vzájemně komunikují a každý zastává určitou funkci. Rozdělení kódu do modulů dle jeho funkce pomáhá k přehlednosti a případné rozšiřitelnosti celé aplikace. Tyto moduly lze rozdělit do čtyřech kategorií: **jádro aplikace**, **vstupní/výstupní moduly**, **nástroje** a **externí moduly**. (Komunikace s nástroji není znázorněna z důvodu přehlednosti.)



Obrázek 4: Schéma architektury aplikace

3.3.1 Jádro aplikace

Moduly jádra aplikace tvoří její hlavní logiku. Obsahují programové smyčky, volají jiné moduly aplikace a mohou být volány ostatními moduly z jádra.

- **Main**:

  Hlavní modul, který se spouští při startu aplikace. Má za úkol inicializaci ostatních modulů a spuštění již zmíněných dvou programových smyček, z nichž jedna slouží pro získávání nových dat a druhá pravidelně kontroluje, zda je stisknuto tlačítko, a pokud ano, postará se o

rozpoznání otázky od uživatele a její zodpovězení. Také slouží pro komunikaci s externím modulem `SpeechCloudClient`.

- **Storage**:

  Centrální modul starající se o práci s daty. Dotazuje se v pravidelných intervalech modulů `API` a `Sensors` na nová data, která zpracuje a pošle modulu `Database` na uložení. Další jeho funkcí je zpracovat významové reprezentace dotazů přicházející z modulu `RequestHandler` a poskytnout mu zpět požadovaná data (získaná pomocí modulu `Database`) ve formě významové reprezentace odpovědi. Slouží také jako buffer uložených dat.

- **RequestHandler**:

  Získává významové reprezentace dotazů od uživatele prostřednictvím externího modulu `SpeechCloudClient`, zpracovává je, odesílá modulu `Storage` a dostává zpět významové reprezentace odpovědí, které pomocí nástroje `Translator` překládá na věty. Utvořené věty poté pošle modulu `SpeechCloudClient`, který je předá uživateli v hlasové podobě. Komunikace s modulem `SpeechCloudClient` probíhá přes modul `Main`.

3.3.2 Vstupní/výstupní moduly

Tyto moduly slouží po komunikaci s externím hardwarem/softwarem. Jsou volány moduly z jádra a nemohou volat jiné moduly.

- **API**:

  Dotazuje se API služby *OpenWeatherMap.org* na data o předpovědi počasí a poskytuje je aplikaci.

- **Sensors**:

  Čte hodnotu teploty a vlhkosti ze senzoru a poskytuje je aplikaci.

- **Database**:

  Stará se o komunikaci s databází MariaDB. Umožňuje ukládání a zpětné čtení předpovědi počasí a záznamu dat ze senzorů. Z databáze získané záznamy ze senzorů poskytuje ve formě maxima, minima a průměru hodnot v záznamech mezi dvěma časovými údaji.

3.3.3 Nástroje

Poskytují nezávislé služby modulům z jádra (nezávislé = výstup je závislý pouze na vstupu a aktuálním čase, nikoliv na stavu aplikace či externího hw/sw). Nemohou volat jiné moduly, které nejsou také nástroji.

- **Translator**:

  Slouží pro vygenerování odpovědi ve formě věty pomocí významových reprezentací dotazu a odpovědi na něj. Také obsahuje předlohy pro oznámení a umožňuje převod mezi různými reprezentacemi data a času.

- **Logger**:

  Modul sloužící k logování oznámení a chybových hlášek aplikace do textových souborů. Více v kapitole 3.8.

3.3.4 Externí moduly

Použité moduly, které nejsou napsané mnou. Nejsou zde uvedeny všechny, pouze ty důležité pro popis fungování aplikace.

- **SpeechCloudClient**:

  Převádí řeč od uživatele pomocí služby *SpeechCloud* na text a provádí syntézu řeči z textu.

3.4 Získávání a ukládání dat

Data jsou získávána ze dvou zdrojů. Předpověd počasí je získávána z API *OpenWeatherMap.org*. Údaje o teplotě a vlhkosti se získávají pomocí senzoru DHT11. Všechna data se ukládají do databáze MariaDB pro jejich trvalé uchování. Databáze obsahuje dvě tabulky: *forcast* a *sensors*. Datum a čas je vždy uložen v UTC v ISO formátu.

3.4.1 OpenWeatherMap API

Každé 2 hodiny se z modulu `API` pošle dotaz na předpověd počasí. Získaná data obsahují údaje o teplotě, tlaku a počasí na následujících 5 dní. Data jsou vzorkována po 3 hodinách (0:00, 3:00, 6:00, ...).

Příklad dat o předpovědi počasí získávaných z API:

```
[
    ...
    {
        'timestamp': '2020-01-24T12:00:00',
        'temperature': 2.0,
        'pressure': 964,
        'weather': 'Clear',
        'description': 'clear sky'
    },
    {
        'timestamp': '2020-01-24T15:00:00',
        'temperature': 0.0,
        'pressure': 963,
        'weather': 'Clouds',
        'description': 'few clouds'
    },
    ...
]
```

Tento seznam se ukládá do databáze do tabulky *forecast*. Jedná se o seznam slovníků, z nichž každý odpovídá jednomu řádku v databázi, a klíče ve slovníku odpovídají sloupcům. Každá hodnota v jednotlivém slovníku se uloží jako hodnota v příslušném řádku a sloupci. Primárním klíčem jednotlivých řádek je pořadí příslušného slovníku v seznamu. Před uložením nové sady předpovědí se odstraní ta předchozí.

Zároveň se předpověď ukládá do bufferu v modulu `Storage`, aby nebylo nutné volat databázi při každém dotazu na předpověď počasí. Při spuštění aplikace jsou do něj nahrána data z databáze a aktualizuje se při každém získání nových dat. Vyprázdní se při ukončení aplikace.

id	timestamp	temperature	pressure	weather	description
1	2020-03-03T15:00:00	5.9	944	Rain	light rain
2	2020-03-03T18:00:00	3.2	946	Snow	light snow
3	2020-03-03T21:00:00	1.7	949	Clouds	overcast clouds
4	2020-03-04T00:00:00	-0.1	950	Clouds	broken clouds
5	2020-03-04T03:00:00	-0.7	951	Clear	clear sky
6	2020-03-04T06:00:00	-0.7	953	Clear	clear sky
7	2020-03-04T09:00:00	3.3	954	Clouds	few clouds

Obrázek 5: Příklad části tabulky *forecast*

3.4.2 Senzor

Data o aktuální teplotě a vlhkosti se získávají každých 10 minut pomocí senzoru fyzicky připojeného k zařízení.

Příklad dat získávaných ze senzoru:

```
{
        'timestamp': '2020-01-23T18:30:00',
        'temperature': 5,
        'humidity': 30
}
```

Tato data ve formátu slovníku se ukládají do databáze do tabulky *sensors*. Každá nová přijatá data se uloží jako řádek tabulky. Jednotlivé klíče slovníku odpovídají sloupcům tabulky. Primárním klíčem je zde pořadí řádku v tabulce.

id	timestamp	temperature	humidity
80	2020-04-17 13:10:00	20	41
81	2020-04-17 13:20:00	21	39
82	2020-04-17 13:30:00	21	39
83	2020-04-17 13:40:00	22	40
84	2020-04-17 13:50:00	21	42
85	2020-04-17 14:00:00	21	41
86	2020-04-17 14:10:00	22	39

Obrázek 6: Příklad části tabulky *sensors*

3.5 Porozumění řečovým dotazům

Porozumění řeči je realizováno pomocí služby *SpeechCloud*, která podle definovaných ABNF gramatik dokáže v řečovém dotazu přiřadit určitým slovům jejich význam a převést tak řečový dotaz na jeho významovou reprezentaci.

Definice jednotlivých ABNF gramatik:

```
$tense = (
    (byl | byla | bylo | byli | byly) {past} |
    (je | jsou) {present} |
    (bude | budou) {future}
);
$subject = (
    (teplota | stupňů | teplo) {temperature} |
    (vlhkost | vlhko) {humidity}
);
$day = (
    (dnes | dneska) {today} |
    zítra {tomorrow} |
    pozítří {tomorrow2} |
    včera {yesterday} |
    předevčírem {yesterday2} |
    pondělí {monday} |
    úterý {tuesday} |
    (středa | středu) {wednesday} |
    čtvrtek {thursday} |
    pátek {friday} |
    (sobota | sobotu) {saturday} |
    (neděle | neděli) {sunday}
);
$time = (
    (noc | noci) {night} |
    (ráno | ránu) {morning} |
    dopoledne {forenoon} |
    odpoledne {afternoon} |
    (večer | večeru) {evening}
);
$condition = (
    (déšť | pršet) {rain} |
    (sněžení | sněžit | sníh) {snow} |
    (slunečno | slunce | jasno) {clear} |
    zataženo {clouds}
);
```

Významová reprezentace dotazu má strukturu slovníku, kde klíčem je jméno pravidla a hodnotou je tag použité alternativy při aplikaci tohoto pravidla. Pro příklad předpokládejme řečový dotaz:

```
Bude zítra odpoledne pršet?
```

Tento dotaz se převede na jeho významovou reprezentaci ve tvaru:

```
{'tense': 'future', 'subject': '', 'day': 'tomorrow',
'time': 'afternoon', 'condition': 'rain'}
```

Údaje, které v takto vytvořené reprezentaci nemají žádné hodnoty, se doplní těmito výchozími hodnotami:

```
{'tense': 'present', 'subject': 'weather', 'day': 'today',
'time': 'day', 'condition': ''}
```

Poté se doplní údaj **type** a určí jeho hodnota. Provede se také korekce údaje **tense** dle hodnoty údaje **day**. Výsledná reprezentace bude tedy ve tvaru:

```
{'tense': 'future', 'type': 'overview', 'subject': 'weather',
'day': 'tomorrow', 'time': 'afternoon', 'condition': 'rain'}
```

Tato významová reprezentace dotazu bude v dalších kapitolách označována jako **dotaz**. Každý z jejích údajů má určitý význam pro následné zpracování a generaci odpovědi:

- **tense**:

  Rozlišuje, zda se jedná o dotaz na minulost (záznamy hodnot), přítomnost (aktuální hodnoty), nebo budoucnost (předpověď).

  - *past*: Minulost
  - *present*: Přítomnost
  - *future*: Budoucnost

- **type**:

  Určuje typ očekávané odpovědi. Zda chceme znát sumarizaci hodnot, celkový přehled, nebo pouze jednu hodnotu.

  - *sum*: Sumarizace hodnot v určitém časovém úseku. Například průměrná teplota za den.
  - *single*: Jedna konkrétní hodnota. Například aktuální teplota.
  - *overview*: Celkový přehled v určitém časovém úseku. Například zítřejší počasí.

- **subject**:

  Předmět dotazu. Rozlišuje, zda se dotaz týká počasí, teploty, či vlhkosti.

  - *weather*: Počasí
  - *temperature*: Teplota
  - *humidity*: Vlhkost

- **day**:

  Den, na který se dotazujeme. Určen relativně vůči aktuálnímu dnu (včera, zítra, v pondělí,...).

  - *yesterday2*: Předevčírem
  - *yesterday*: Včera
  - *today*: Dnes
  - *tomorrow*: Zítra
  - *tomorrow2*: Pozítří
  - *monday, tuesday, ...* : Pondělí, úterý, ...

- **time**:

  Konkrétní denní doba, která nás zajímá.

  - *day*: Den (6:30 až 19:30)
  - *night*: Noc (19:30 až 6:30)
  - *morning*: Ráno (6:30 až 10:30)
  - *forenoon*: Dopoledne (8:30 až 12:30)
  - *afternoon*: Odpoledne (12:30 až 17:30)
  - *evening*: Večer (16:30 až 19:30)

- **condition**:

  Konkrétní počasí, o kterém nás zajímá, zda nastane.

  - *clear*: Jasno
  - *clouds*: Zataženo
  - *snow*: Sněžení
  - *rain*: Déšť (zahrnuje i bouřky a mrholení)

3.6 Určení data a času z dotazu

Dotaz obsahuje pouze relativní a samy o sobě nepříliš vypovídající informace o datu a času (včera, v úterý, atd.). Proto je třeba provést převod na konkrétní časový údaj.

Příklad:

Uvažujme **dotaz** z předchozí kapitoly a aktuální datum a čas v místní časové zóně (ISO formát):

```
2020-06-23T18:22:34
```

Nejprve se datum a čas posune o počet dní, který odpovídá údaji **day**. Pokud je den určen názvem dne v týdnu, nalezne se nejbližší odpovídající den (v budoucnu či v minulosti v závislosti na údaji **tense**). V tomto případě `tomorrow` odpovídá posunu `+1`. Dostaneme tedy datum:

```
2020-06-24
```

Dále se zjistí počátek a konec časového úseku, který odpovídá údaji **time**. V tomto případě **afternoon** odpovídá úseku:

```
start: 12:30
end: 17:30
```

Tento úsek se převede na odpovídající čas v UTC. Dle systémového času se určí odchylka oproti UTC, která se odečte od počátku i konce úseku. Pokud předpokládáme místní časové pásmo GMT+1 a letní čas (celková odchylka +2 od UTC), musí se úsek posunout o dvě hodiny zpět:

```
start: 10:30
end: 15:30
```

Pokud se jedná o dotaz na záznamy ze senzorů, sloučí se oba časové údaje se dříve zjištěným datem, čímž se získá začátek a konec požadovaného úseku.

Pro předpověď počasí je nutné provést ještě jeden krok, a to nalezení všech násobků periody uložených dat (3 hodiny), které spadají do úseku z předešlého kroku. V tomto případě máme násobky dva:

```
12:00, 15:00
```

Tyto násobky se sloučí s datem zjištěným dříve:

```
['2020-06-24T12:00:00', '2020-06-24T15:00:00']
```

Čímž se získá pole konkrétních dat a časů v UTC, dle kterého můžeme vyhledat data v databázi.

3.7 Zpracování dotazu a vytvoření odpovědi

Při zpracování **dotazu** je vygenerována významová reprezentace odpovědi, která obsahuje uživatelem požadované informace. Zpracování **dotazu** i formát odpovědi se lehce liší podle toho o jaký druh **dotazu** se jedná. Odpověď ve formě věty je generována dosazením do předdefinovaných šablon. Aplikace dokáže zpracovat tři typy **dotazů**: dotaz na předpověď počasí, na záznamy ze senzoru a na aktuální údaje ze senzoru.

3.7.1 Předpověď počasí

Pokud uživatel požaduje data týkající se předpovědi počasí, tedy pokud pro **dotaz** platí: **tense** = *future*, **type** = *overview* a **subject** = *weather*, probíhá zpracování následujícím způsobem:

Pro příklad předpokládejme otázku:

```
Bude zítra pršet?
```

A z ní vytvořený **dotaz**:

```
{'tense': 'future', 'type': 'overview', 'subject': 'weather',
'day': 'tomorrow', 'time': 'day', 'condition': 'rain'}
```

A aktuální datum a čas:

```
2020-01-29T14:00:00
```

Nejprve se určí pole konkrétních dat a časů v UTC, které odpovídají danému **dotazu** (viz přechozí kapitola):

```
['2020-01-30T06:00:00', '2020-01-30T09:00:00', '2020-01-30T12:00:00',
 '2020-01-30T15:00:00', '2020-01-30T18:00:00']
```

Poté se z databáze získají data o předpovědi odpovídající těmto časovým údajům. Z těchto dat se vypočte minimální a maximální teplota (v °C):

```
min: 2.0
max: 4.0
```

Dále se sestaví přehled předpovědi počasí. Jedná se o seskupení časových údajů podle druhů počasí:

```
[
    {'weather': 'Snow', 'timestamps': ['2020-01-30T06:00:00']},
    {'weather': 'Rain', 'timestamps': ['2020-01-30T09:00:00']},
    {'weather': 'Clouds', 'timestamps': ['2020-01-30T12:00:00',
     '2020-01-30T15:00:00', '2020-01-30T18:00:00']}
]
```

Pokud existuje v dotazu podmínka (údaj **condition**), určí se tzv. `condition_index`. Ten je roven indexu druhu počasí daného podmínkou v poli přehledu. Pokud se druh počasí v přehledu nevyskytuje, je roven -1.

V tomto případě je roven 1, jelikož *rain* je v přehledu na indexu 1.

Výsledná významová reprezentace odpovědi:

```
{
    'temperature': {
        'min': 2.0,
        'max': 4.0
    },
    'weather': [
        {
            'weather': 'Snow',
            'timestamps': [
                '2020-01-30T06:00:00'
            ]
        },
        {
            'weather': 'Rain',
            'timestamps': [
                '2020-01-30T09:00:00'
            ]
        },
        {
            'weather': 'Clouds',
            'timestamps': [
                '2020-01-30T12:00:00',
                '2020-01-30T15:00:00',
                '2020-01-30T18:00:00'
            ]
        }
    ],
    'condition_index': 1
}
```

Poté je potřeba vytvořit odpověď ve formě věty. Nejprve se datum a čas převede na úsek daný pouze hodinami:

```
['6.', '9.', '12. až 18.']
```

Tyto úseky se vyjádří v místní časové zóně UTC+1:

```
['7.', '10.', '13. až 19.']
```

Nakonec se pomocí údajů z **dotazu** a odpovědi vytvoří reakce na podmínku:

```
Ano, zítra očekávejte déšť kolem 10. hodiny.
```

Celkový přehled počasí (vyjma počasí v podmínce):

```
Kolem 7. hodiny očekávejte sněžení.
Kolem 13. až 19. hodiny očekávejte zataženo.
```

A údaje o teplotě:

```
Teplota se bude pohybovat mezi hodnotami 2 až 4 stupně.
```

3.7.2 Záznamy ze senzoru

Pokud uživatel požaduje záznamy ze senzorů, tedy pokud pro **dotaz** platí: **tense** = *past* a **type** = *sum*, probíhá zpracování následujícím způsobem:

Pro příklad předpokládejme otázku:

```
Jaká byla včera odpoledne teplota?
```

A z ní vytvořený **dotaz**:

```
{'tense': 'past', 'type': 'sum', 'subject': 'temperature',
'day': 'yesterday', 'time': 'afternoon', 'condition': ''}
```

A aktuální datum a čas:

```
2020-01-30T14:00:00
```

Nejprve se určí konkrétní datum a čas v UTC počáteční a koncové hodnoty časového úseku daného **dotazu** (viz přechozí kapitola):

```
start: 2020-01-29T12:30:00
end: 2020-01-29T17:30:00
```

Poté se pošle dotaz na databázi pro získání minimální, maximální a průměrné teploty (ve °C) v tomto časovém úseku:

```
min: 1.0
max: 9.0
avg: 5.5
```

Výsledná významová reprezentace odpovědi:

```
{
    'temperature': {
        'min': 1.0,
        'max': 9.0,
        'avg': 5.5
    }
}
```

Nakonec se pomocí údajů z **dotazu** a odpovědi vygeneruje odpověď ve formě věty:

```
Teplota se včera odpoledne pohybovala mezi hodnotami 1 až 9 stupňů.
Průměrná teplota byla 5,5 stupňů.
```

3.7.3 Aktuální údaje ze senzoru

Pokud uživatel požaduje aktuální hodnoty ze senzorů, tedy pokud pro **dotaz** platí: **tense** = *present* a **type** = *single*, probíhá zpracování následujícím způsobem:

Pro příklad předpokládejme otázku:

```
Jaká je teplota?
```

A z ní vytvořený **dotaz**:

```
{'tense': 'present', 'type': 'single', 'subject': 'temperature',
'day': 'today', 'time': 'day', 'condition': ''}
```

Nejprve se získá aktuální hodnota, ze které se vytvoří významová reprezentace odpovědi:

```
{'temperature': 7.0}
```

A ta se převede do formy věty:

```
Aktuální teplota je 7 stupňů.
```

3.8 Logování a chybová hlášení

Modul `logger` zajišťuje logování chování aplikace do textových souborů, které se ukládají do složky *logs*. Tyto logy se zároveň vypisují i do konzole. Textové soubory jsou pojmenovány ve formátu:

```
{rok}-{měsíc}-{den}.txt
```

Každá řádka tohoto soubory představuje jednotlivý *log*, které se skládá ze dvou částí: data a času pořízení a obsah logu. Formát logu tedy vypadá takto:

```
[{rok}-{měsíc}-{den} {hodiny}:{minuty}:{sekundy}] {obsah logu}
```

Aplikace rozlišuje dva typy logů: **oznámení** a **chybové hlášení**. Oznámení pouze informují, co se v aplikaci stalo. Chybová hlášení upozorňují na případy, kdy nebylo možné zpracovat **dotaz** z důvodu nedostupnosti dat, a na možné chyby při pokusu získat nová data.

Seznam logovaných událostí a příklady obsahu logů:

- **Oznámení:**
  - Aktualizace předpovědi počasí

    ```
    Předpověď počasí byla aktualizována.
    ```

  - Záznam nových údajů z senzoru

    ```
    Zaznamenány údaje ze senzoru - teplota: 7 stupňů, vlhkost: 25.
    ```

  - Vygenerování odpovědi na otázku uživatele

    ```
    Odpověď: "Aktuální teplota je 18 stupňů."
    ```

- **Chybová hlášení:**
  - Neexistence záznamů ze senzoru v daném čase

    ```
    CHYBA: Nebyly nalezeny žádné záznamy v čase
    od '2020-07-31 10:30:00' do '2020-07-31 15:30:00'
    ```

  - Chyba při čtení dat ze senzoru

    ```
    CHYBA: Nepodařilo se načíst data ze senzoru
    ```

- Chyba při stažení předpovědi počasí z API

```
CHYBA: Nepodařilo se stáhnout předpověď z API (status code: 404)
```

- Nedostupné připojení k internetu

```
CHYBA: Připojení k internetu není dostupné
```

- Chyba připojení k databázi

```
CHYBA: Připojení k databázi se nezdařilo
```

Ukázka logů v souboru:

```
[2020-08-07 11:05:47] Odpověď: "Aktuální teplota je 24.0 stupňů."
[2020-08-07 11:08:27] Předpověď počasí byla aktualizována
[2020-08-07 11:08:44] Odpověď: "Kolem 7. až 13. hodiny očekávejte slunečno. Kolem 16. až 19.
hodiny očekávejte zataženo. Teplota se bude pohybovat mezi hodnotami 22 až 29 stupňů."
[2020-08-07 11:09:13] CHYBA: Nebyly nalezeny žádné záznamy v čase od '2020-08-06 05:30:00' do
'2020-08-06 18:30:00'
[2020-08-07 11:09:13] Odpověď: "Nebyly nalezeny žádné záznamy"
[2020-08-07 11:09:33] Odpověď: "Aktuální teplota je 24.0 stupňů."
[2020-08-07 11:09:53] Odpověď: "Kolem 19. hodiny očekávejte déšť. Teplota se bude pohybovat
kolem hodnoty 22 stupňů."
[2020-08-07 11:10:05] Zaznamenány údaje ze senzoru - teplota: 25.0 °C, vlhkost: 31.0
[2020-08-07 11:10:33] Odpověď: "Ne, v neděli ráno neočekávejte déšť. Kolem 7. až 10. hodiny
očekávejte slunečno. Teplota se bude pohybovat mezi hodnotami 22 až 27 stupňů."
[2020-08-07 11:13:30] Předpověď počasí byla aktualizována
[2020-08-07 11:13:55] Odpověď: "Nevím. Stačí se podívat z okna."
[2020-08-07 11:14:11] Odpověď: "Počasí z minulosti není zaznamenáváno. Pouze teplota a vlhkost."
```

Obrázek 7: Ukázka logů v souboru 2020-08-07.log

3.9 Příklady hlasového dialogu

Zde jsou uvedeny příklady hlasové dialogu uživatele se systémem (odpovědi na uživatelovi otázky).

Uživatel: Jaké bude zítra počasí?
Systém: Kolem 8. až 17. hodiny očekávejte zataženo. Teplota se bude pohybovat mezi hodnotami 22 až 28 stupňů.

Uživatel: Jak bude v pondělí ráno?
Systém: Kolem 8. hodiny očekávejte zataženo. Teplota se bude pohybovat kolem hodnoty 22 stupňů.

Uživatel: Bude zítra odpoledne pršet?
Systém: Ne, zítra odpoledne neočekávejte déšť. Kolem 14. až 17. hodiny očekávejte zataženo. Teplota se bude pohybovat mezi hodnotami 27 až 28 stupňů.

Uživatel: Jaká bude v úterý dopoledne teplota?
Systém: Teplota se v úterý dopoledne bude pohybovat kolem hodnoty 24 stupňů.

Uživatel: Jaká byla předevčírem odpoledne vlhkost?
Systém: Vlhkost se předevčírem odpoledne pohybovala mezi hodnotami 25 až 34. Průměrná vlhkost byla 31.

Uživatel: Jaká je venku teplota?
Systém: Aktuální teplota je 18 stupňů.

Uživatel: Jaké bylo včera počasí?
Systém: Počasí z minulosti není zaznamenáváno. Pouze teplota a vlhkost.

Uživatel: Jak bude v pátek?
Systém: Nebyly nalezeny žádné záznamy.

4 Závěr

V této práci byl popsán princip fungování jednoduché hlasově ovládané meteostanice. V první části byla představena teorie za jednotlivými součástmi projektu. Tato teorie zahrnovala hlasové dialogové systémy včetně rozpoznávání a syntézy řeči. Zmíněna byla i v projektu využívána platforma *SpeechCloud*.

Dále byla popsána architektura REST API včetně služby *OpenWeatherMap*, která poskytuje informace o počasí pomocí tohoto rozhraní. Byla také představena problematika databázových systémů, především relačních databází a jimi používaného programovacího jazyka SQL. Jako poslední byla v teoretické části představena historie a specifikace jednočipového mikropočítače Raspberry Pi, na kterém běží software vytvořené meteostanice.

V praktické části práce byla popsána celková architektura jak hardwarové tak softwarové části projektu. Obsahuje popis jednotlivých procesů v softwaru, především postup získávání potřebných dat ze senzoru a služby *OpenWeatherMap*, ukládání/čtení těchto dat do/z databáze, porozumění dotazům od uživatele a generování srozumitelné odpovědi na tyto dotazy.

Tato práce by mohla sloužit jako základ pro mnohem komplexnější hlasově ovládanou meteostanici s detekcí více veličin a vlastní předpovědí počasí, případně také pro užitečnou součást chytré domácnosti poskytující informace o předpovědi počasí nebo teplotě a vlhkosti venku či v místnosti.

Seznam použitých zdrojů

1. PSUTKA, Josef. Přednáška: Analýza a zpracování řečového signálu, parametrizace řeči. *Úvod do strojového vnímání prostředí*. 2020.
2. IRCING, Pavel. Přednáška: Rozpoznávání řeči, akustické a jazykové modelování. *Úvod do strojového vnímání prostředí*. 2020.
3. MATOUŠEK, Jindřich. Přednáška: Zpracování textu a syntéza řeči. *Úvod do strojového vnímání prostředí*. 2020.
4. PSUTKA, Josef. *Mluvíme s počítačem česky*. Praha: Academia, 2006. ISBN 80-200-1309-1.
5. ŠVEC, Jan. Přednáška: Hlasové dialogové systémy. *Úvod do strojového vnímání prostředí*. 2020.
6. ŠVEC, Jan. *SpeechCloud API*. 2016. Dostupné z: `https://docs.google.com/presentation/d/1bV4nedeZOjfgwbJIbJf7tR8GpZfMCAWoywfPQGJKVPA`.
7. Speech Recognition Grammar Specification Version 1.0. *W3C*. [online]. Dostupné z: `https://www.w3.org/TR/speech-grammar`.
8. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures* [online]. 2000. Dostupné z: `https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`.
9. Usage statistics of web servers. *W3Techs*. [online]. Dostupné z: `https://w3techs.com/technologies/overview/web_server`.
10. HTTP request methods. *Mozilla*. [online]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods`.
11. HTTP Status Codes. *restfulapi.net*. [online]. Dostupné z: `https://restfulapi.net/http-status-codes/`.
12. Weather API. *OpenWeatherMap*. [online]. Dostupné z: `https://openweathermap.org/api`.
13. Technology. *OpenWeatherMap*. [online]. Dostupné z: `https://openweathermap.org/technology`.
14. What is Database? *Guru99*. [online]. Dostupné z: `https://www.guru99.com/introduction-to-database-sql.html`.
15. A Relational Database Overview. *Oracle*. [online]. Dostupné z: `https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html`.
16. KREIBICH, Jay. *Using SQLite* [online]. O'Reilly Media, Inc., 2010. Dostupné z: `https://books.google.cz/books?id=HFIM47wp0X0C`.
17. CHATHAM, Mark. *Structured Query Language By Example - Volume I: Data Query Language* [online]. 2012. Dostupné z: `https://books.google.cz/books?id=64MBBAAAQBAJ`.

18. About. *MariaDB Foundation*. [online]. Dostupné z: https://mariadb.org/about.

19. Distributions Which Include MariaDB. *MariaDB Foundation*. [online]. Dostupné z: https://mariadb.com/kb/en/distributions-which-include-mariadb.

20. Structure of the Registry. *Microsoft*. [online]. Dostupné z: https://docs.microsoft.com/cs-cz/windows/win32/sysinfo/structure-of-the-registry.

21. CELLAN-JONES, Rory. A 15 pound computer to inspire young programmers [online]. *BBC*. 2011. Dostupné z: https://www.bbc.co.uk/blogs/thereporters/rorycellanjones/2011/05/a_15_computer_to_inspire_young.html.

22. Products. *Raspberry Pi Foundation*. [online]. Dostupné z: https://www.raspberrypi.org/products.

23. Documentation - Hardware. *Raspberry Pi Foundation*. [online]. Dostupné z: https://www.raspberrypi.org/documentation/hardware/raspberrypi.

24. Documentation - GPIO. *Raspberry Pi Foundation*. [online]. Dostupné z: https://www.raspberrypi.org/documentation/usage/gpio.

25. Documentation - Raspberry Pi OS. *Raspberry Pi Foundation*. [online]. Dostupné z: https://www.raspberrypi.org/documentation/raspbian.

26. Documentation - Usage. *Raspberry Pi Foundation*. [online]. Dostupné z: https://www.raspberrypi.org/documentation/usage.