Univerzita Hradec Králové

Fakulta informatiky a managementu
Katedra informačních technologií

Využití hlasových asistentů v chytrých domácnostech

Bakalářská práce

Autor: David Tláskal
Studijní obor: Aplikovaná informatika (ai3-p)
Vedoucí práce: Mgr. Hana Rohrová
Odborný konzultant: Ing. Jan Štěpán

Hradec Králové duben 2021

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 29. 4. 2021 David Tláskal

Poděkování

Děkuji vedoucí bakalářské práce Mgr. Haně Rohrové za odborné vedení, za výpomoc a rady při zpracování této práce. Dále také děkuji Ing. Janu Štěpánovi za rady ohledně odborných částí práce a za zapůjčení dedikovaných zařízení hlasových asistentů.

Abstrakt

Práce se zabývá základními principy vybraných hlasových asistentů, mezi které patří například převod hlasu na text, zpracování přirozeného jazyka a převod textu na hlas. Kromě těchto základních principů se práce dále zabývá konverzačním API (Application Programming Interface) a API pro zařízení chytrých domácností.

Cílem této práce je analyzovat jednotlivá API vybraných hlasových asistentů a na základě této analýzy navrhnou a vytvořit aplikaci, která bude sloužit jako prostředník mezi hlasovými asistenty a zařízeními chytrých domácností.

Abstract

Title: Evaluation of voice assistants in smart homes

The thesis deals with the basic principles of selected voice assistants, which include, for example, the conversion of voice to text, natural language processing and the conversion of text to voice. In addition to these basic principles, the work also deals with the conversational API (Application Programming Interface) and API for smart home devices.

The aim of this work is to analyze the individual APIs of selected voice assistants and based on this analysis to design and create an application that will serve as an intermediary between voice assistants and smart home devices.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V současné době se hlasové asistenty staly trendem mezi uživatelskými rozhraními. Hlavním důvodem je jednoduchost interakce mezi samotným asistentem a uživatelem. Na rozdíl od tradičních uživatelských rozhraní, se hlasové asistenty ovládají pomocí vyslovených příkazů. Výhodou tohoto přístupu je, že hlasový asistent může plnit požadavky uživatele i když je uživatel zaneprázdněn jinou činností. Snahou hlasových asistentů je imitovat přirozenou mezilidskou konverzaci a podle toho je i navrženo konverzační rozhraní jednotlivých asistentů.

Dalším z trendů dnešní doby se stává IoT (Internet Of Things). IoT se zabývá chytrými zařízeními s internetovou konektivitou. Tato zařízení mohou být využívána například v tzv. chytrých domácnostech. Vzhledem k jejich internetové konektivitě je možné zařízení propojit pomocí aplikací rozšiřujících funkcionality hlasových asistentů se samotnými hlasovými asistenty a tím umožnit komfortní ovládání těchto zařízení.

Aby mohl výrobce IoT zařízení své produkty propojit s hlasovými asistenty, je nutno navrhnout a vytvořit aplikaci, která bude komunikovat s některým z API jednotlivých asistentů. Služby vybraných hlasových asistentů poskytují dvě API pro integraci takových aplikací:

1. Konverzační API [1, 2]
2. Smart Home API [3, 4]

Každé z těchto rozhraní má své limity. Proto je nezbytné se v jednotlivých rozhraních zorientovat a následně vybrat, které API je nejvhodnější. Pokud ani jedno z těchto API nesplňuje nároky, je možná kombinace obou rozhraní. Ovšem vzhledem k velikosti jednotlivých API jejich kombinování výrazně přispívá ke komplexnosti aplikace.

Kromě samotného rozhraní asistenta je nutno navrhnout a implementovat autentizační rozhraní pro uživatele vytvořené aplikace. Toto rozhraní slouží k identifikaci uživatele a k bezpečnému přenosu dat mezi aplikací a API vybraných hlasových asistentů. K implementaci takového rozhraní je používán OAuth2.

Dalším klíčovým aspektem je komunikace mezi IoT zařízeními a samotnou aplikací. V IoT je velmi často používán komunikační protokol MQTT (Message Queuing Telemetry Transport). Tento protokol

je založen na *Client/Broker* a *Publish/Subscribe* architektuře. Tento protokol je velmi efektivní a nenáročný. Díky tomu může být použit i na jednodušších zařízeních.

2 Cíl práce

Cílem práce je objasnit základní principy vybraných hlasových asistentů, tzn. záznam řeči, převod řeči na text, analýza textu, zpracování požadavku a tvorba odpovědi. Dále také prozkoumat možnosti propojení vybraných hlasových asistentů se službami třetích stran a následně navrhnout a vytvořit službu, která umožní propojení chytrých zařízení s touto službou a následně i s těmito hlasovými asistenty.

3 Zpracování řeči a přirozeného jazyka

Aby mohl hlasový asistent reagovat na hlasové povely, musí tyto povely zaznamenat, zpracovat a vygenerovat odpověď. Před samotným zpracováním hlasových povelů je nutné analyzovat řeč a převést ji na text a z tohoto textu následně získat potřebné informace na základě kterých bude generována odpověď.

3.1 Řeč

Řeč je nejčastěji zvukový projev člověka a slouží především ke vzájemnému dorozumívání. Řeč se skládá ze slov, která tvoří jakýsi lexikální systém. Skládání slov je závislé na gramatických pravidlech daného jazyka. Slova se skládají z hlásek, které tvoří fonémický systém daného jazyka. [5] Porozumění této struktuře je klíčové pro samotné zpracování řeči.

3.1.1 Fonémy

Fonémy jsou nejmenší rozlišitelnou částí řeči. Oproti alofonům jsou fonémy čistě abstraktním prvkem, který pouze napomáhá k porozumění řeči. Každá řeč má svou vlastní sadu fonémů. Fonémy se označují dvojicí dopředných lomítek, např. /t/. Vzhledem k tomu, že foném je nejmenší rozlišitelná část řeči, tak může i jediná změna fonému, změnit význam celého slova. Pokud například změníme foném /r/ v anglickém slově *rat* tj. krysa na foném /b/, vznikne slovo *bat* tj. netopýr. [6]

3.1.2 Alofony

Alafony jsou akustickou realizací fonémů, tudíž nejsou abstraktním prvkem mluvené řeči. Každý foném může mít jeden, nebo více alofonů. Záměna alfonů oproti záměně fonémů nemusí nutně změnit význam slova. To jaký je použit alofon obvykle záleží na kontextu, a výskytu okolních alofonů. Příkladem může být ztráta znělosti souhlásek na konci slov v českém jazyce. Například /f/ se v přirozených českých slovech vyskytuje zřídka a obvykle pouze jako neznělá realizace /v/, například ve slově *ostrov* [ostrof] a v ostatních tvarech slova se vyslovuje [v] tj. ostrovy [ostrovi]. Vliv na výskyt alofonů může mít například i dialekt, nebo fyziologie určitého jedince. [6] Kromě rozpoznávání řeči jsou alofony důležitým prvkem i při její syntetizaci.

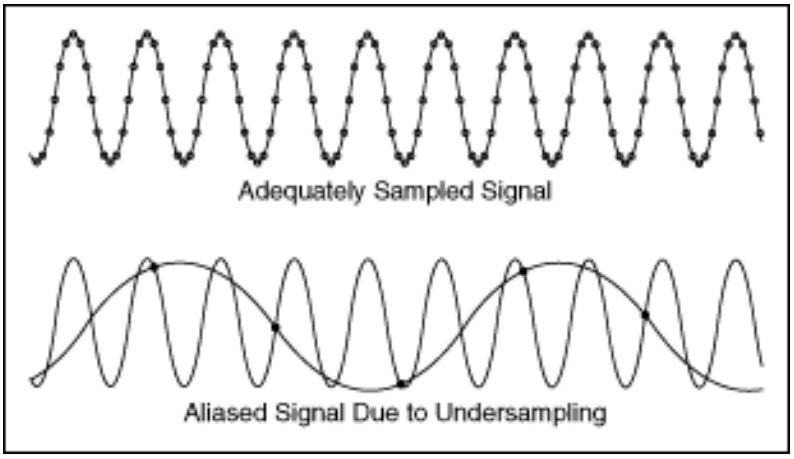
3.2 Digitální záznam řeči

Před samotným převodem hlasu na text je nutné hlas zachytit. K tomuto procesu se používá mikrofon, analogově digitální převodník a popřípadě kodek. [5]

3.2.1 Zachycení

Akustický signál je tvořen z vln. Tyto akustické vlny jsou zachyceny pomocí mikrofonu. Cílem mikrofonu je převést tyto vlny na elektrickou veličinu tj. napětí, nebo proud. Následně je tato veličina dále upravována. Jednou z těchto úprav může být například zesílení, nebo odstranění šumu. Po těchto úpravách je veličina připravena k dalšímu kroku. [5]

3.2.2 ADC



Obrázek 1: Vzorkování analogového signálu a ukázka aliasu. Zdroj:[7]

ADC (Analog to Digital Converter) je hardwarový prvek jehož úkolem je převádět analogový signál na digitální. Samotný převod lze rozdělit do dvou kroků. Prvním krokem je tzv. vzorkování a druhým je tzv. kvantování.

Úkolem vzorkování je ze spojitého signálu periodicky odebírat vzorky. Příklad vzorkování je možné vidět v horní polovině obrázku 1. Frekvence vzorkování musí být minimálně dvojnásobná, než je frekvence samotného signálu. Pokud by tomu tak nebylo, docházelo by k aliasu. Příklad aliasu je možné vidět ve spodní polovině obrázku 1. Frekvence signálu běžné řeči je v rozmezí 3kHz až 10kHz. To znamená, že vzorkovací frekvence řeči musí být minimálně 20kHz. [7]

Cílem kvantování je tyto vzorky zařadit do určitých hladin. Tyto hladiny jsou limitovány samotným hardwarem ADC. Pro kvantování řeči je potřeba použít minimálně 11-12bit kvantování. [5] Běžně používaná rozlišení jsou 8bit (256 hladin), 16bit (65536 hladin) a 24bit (16,8 milionů hladin). Výsledný digitální signál je aproximací jeho analogového předchůdce.

3.2.3 Kodek

Po úspěšném převedení analogového signálu na digitální, je potřeba objem dat digitálního signálu zmenšit. Jak už samotný proces převodu napovídá, výsledný digitální signál může být poměrně veliký. Pokud například vezmeme vzorkovací frekvenci 20kHz a 16 bitové kvantování, vyjde nám 320kbit/s. Pro zmenšení digitálního záznamu se používají kodeky. Kodeky jsou hardwarové, nebo softwarové prvky, které podle jim specifických postupů zmenšují a ukládají nebo čtou daný digitální signál. [5]

3.3 Získávání informací z řeči

Pro účely rozpoznávání řeči, se nejčastěji používá krátkodobá analýza. Zvukový signál obsahuje spoustu nadbytečných informací a cílem této analýzy je snížení tohoto počtu informací. Všechny typy metod krátkodobé analýzy jsou aplikovány na úseky signálu, které jsou v rozmezí od 10 do 30 milisekund. Proto se nejprve provede segmentace signálu na jednotlivé rámce, na které se následně aplikují metody krátkodobé analýzy. Tímto způsobem získáme pro každý rámec skalár (jeden příznak), nebo vektor (více příznaků). Důležitou vlastností je, aby výsledek této analýzy byl jednoznačný pro každý typ signálu a byl co nejméně ovlivněn okolními podmínkami a množstvím šumu. Obecně tuto analýzu můžeme rozdělit do dvou oblastí, kdy se jedna oblast zabývá časovou doménou a druhá frekvenční. [5]

3.3.1 Časová doména krátkodobé analýzy

Oblast krátkodobé analýzy je oproti té frekvenční poměrně nenáročná na výpočetní výkon, ale nabídne pouze základní informace. Mezi její hlavní využití patří rozhodování, zdali je signál řeč, nebo šum. Dále také znělost a neznělost řeči a množství příznaků následně využitých ve frekvenční doméně krátkodobé analýzy. Mezi příznaky získané

z časového průběhu patří například krátkodobá intenzita, krátkodobá funkce středního počtu průchodu amplitudy signálu nulou a nebo krátkodobá autokorelační funkce. [5]

3.3.2 Frekvenční doména krátkodobé analýzy

Mluvená řeč je zastoupena spektrem frekvencí, proto se jedná o krátkodobou spektrální analýzu. Základem většiny metod pro zpracování frekvenční domény je Fourierova transformace. Vzhledem k tomu, že signál v jednotlivých rámcích je diskrétní, je použita diskrétní verze Fourierovy transformace. I po tomto procesu toto spektrum řeči obsahuje spoustu nadbytečných informací. Proto je toto spektrum rozděleno do několika pásem a každé pásmo je zpracováno jednotlivě a vypočítáno tzv. kepstrum. Kepstrální analýza je metoda umožňující ze signálu řeči oddělit parametry buzení a hlasového ústrojí. Parametry buzení mohou být využity například k identifikování fonémů. [5]

3.4 Rozpoznání řeči

Po získání akustických informací z řeči, je možné části řeči klasifikovat a rozpoznat. V historii byl často používaný algoritmus založený na DTW (Dynamic Time Warping) a v dnešní době se stal velmi populární algoritmus založený na HMM (Hidden Markov Models).

3.4.1 DTW algoritmus

Dynamic Time Warping, je algoritmus rozpoznávání řeči založený na porovnávání akustických šablon. DTW měří podobnosti mezi dvěma sekvencemi bez ohledu na čas nebo rychlost obou sekvencí. Toto je velmi důležitá vlastnost, protože řečník může stejné slovo vyslovit rychleji či pomaleji s ohledem na kontext.

Vzhledem k tomu, že tento systém nerozlišuje jednotlivé mluvčí, musí být akustické šablony namluveny několika mluvčími s rozdílnými hlasy, věky, dialekty a následně seskupeny do svazků pro jednotlivá slova. Tato skutečnost napovídá tomu, že tento princip rozpoznávání hlasu může být značně náročný na úložný prostor.

Poslední krok DTW algoritmu je takzvané rozhodovací pravidlo, které vybírá šablonu nejblíže podobnou slovu mluvenému. K tomuto rozhodování se nejčastěji používají variace algoritmu nejbližších sousedů.

Dále byl algoritmus DTW rozšířen pro rozpoznávání frází. Princip rozpoznávání slov zůstává stejný, pro výsledná slova je nadále vyhledána fráze ze slovníku spojených slov.

Další nadstavbou DTW algoritmu je Bridle algoritmus. Tento algoritmus postupně tvoří stromovou strukturu možných slov podle skóre dosaženého z DTW. Tato stromová struktura je výsledně použita při skládání fráze pro rozhodování které slovo se hodí nejvíce. [5]

3.4.2 HMM algoritmus

HMM (Hidden Markov Models), neboli Markovovy modely jsou statistické modely se skrytými stavy. Markovův algoritmus je dopředně zpětný algoritmus. Tzn, že nynější stav modelu není viditelný pozorovateli. Viditelný je pouze výstup tohoto procesu. Přesto, že jsou parametry modelu předem známy a definovány, má tento model přívlastek skrytý a to právě z toho důvodu, že se vztahuje na vnitřní stavy modelu a ne na parametry. Posloupnost výstupu, který vygeneroval Markovův model vypovídá o vnitřních stavech modelu. Zpětně je možno určit na základě výstupu pravděpodobnost s jakou je tento výstup pravdivý. Tyto modely jsou vhodné pro rozpoznávání časových vzorů, proto se stejně jako DTW používají při rozpoznávání řeči. [8]

Při rozpoznávání řeči pomocí skrytých Markovových modelů, je zvuk rozdělen do časových oken, které obsahují jednotlivé fonémy. V prvním kroku se pomocí pravděpodobnosti určí o jakou fonému se nejspíše jedná. V dalším kroku je zjištěna pravděpodobnost, zdali mohou být určené fonémy vedle sebe. Pokud nemohou, tak se algoritmus vrátí k předešlému kroku a pokusí se chybu opravit a takto se z foném složí slova. V dalším kroku algoritmus porovnává pravděpodobnosti, zdali mohou být tato slova vedle sebe a pokud ne, tak se opět vrací o krok zpět. Čím je kontext větší, tím přesnější je i samotné rozpoznávání a proto je možné pozorovat i změnu slov na začátku věty až po tom, co je rozpoznán její konec. Tento algoritmu je oproti DTW poměrně nenáročný na úložný prostor. [5]

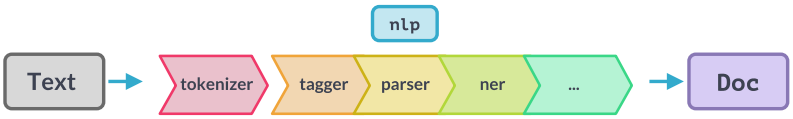
3.4.3 HMM algoritmus a neuronové sítě

Rozpoznávání hlasu pomocí skrytých Markovových modelů je neflexibilní proces. To znamená, že dokáže rozpoznat pouze předem určené prvky. Z tohoto důvodu, se často používají neuronové sítě, které využívají uzly a váhy mezi nimi k určení výsledku. Prázdná neuronová

síť má velkou chybovost, protože nemá správně nastavené váhy mezi jednotlivými uzly. Neuronová síť tyto váhy upravuje sama pomocí trénovacích dat a zpětné vazbě o chybovosti předešlého určení. [5]

3.5 Zpracování přirozeného jazyka

Přirozený jazyk je jakýkoli jazyk vyvinutý lidmi a používaný v běžné komunikaci. Vzhledem k různorodosti jazyků a jejich forem, je složité jazyky zpracovat strojově. Snahou technik pro zpracování přirozeného jazyka je, aby stroje částečně porozuměly přirozené řeči. Toto porozumění je závislé na předem vytvořených korpusech pro daný jazyk, popřípadě pro daný kontext.



Obrázek 2: spaCy NLP pipeline. Zdroj:[9]

Zpracování přirozeného jazyka je proces, který se běžně skládá z několika částí. Skládáním těchto částí se tvoří tzv. *pipeline* (viz. obrázek 2). S ohledem na řešenou problematiku se tento pipeline může měnit. V následujících částech práce jsou popsány základní a vždy využívané prvky.

Tabulka 1: Zpracovaná věta přirozeného jazyka. Zdroj: [autor]

Text	Lemma	Pos	Tag	Entity
In	in	ADP	IN	
2016	2016	NUM	CD	DATE
,	,	PUNCT	,	
David	David	PROPN	NNP	PERSON
visited	visit	VERB	VBD	
the	the	DET	DT	LOC
Eiffel	Eiffel	PROPN	NNP	LOC
Tower	Tower	PROPN	NNP	LOC
,	,	PUNCT	,	
which	which	DET	WDT	
is	be	AUX	VBZ	
located	locate	VERB	VBN	
in	in	ADP	IN	
Paris	Paris	PROPN	NNP	GPE
.	.	PUNCT	.	

3.5.1 Tokenizace

Základním prvkem psaného jazyka jsou slova a interpunkce. Tyto základní prvky jsou v kontextu zpracování přirozeného jazyka nazývány tokeny a jedním z prvních kroků při tomto zpracování je kontinuální text rozdělit na tyto tokeny. Dvě základní varianty jak toho docílit jsou white space tokenization a regular expression tokenization. Výsledek tokenizace je možné vidět ve sloupci Text tabulky 1.

White space tokenization je založena na jednoduchém předpokladu, že jsou jednotlivé tokeny odděleny jedním z white space znaků. Příkladem běžně používaných white space znaků může být například mezeza, tabulátor, nový řádek. Vzhledem k jednoduchosti pravidel, se tento algoritmus může dopouštět chyb.

Regular expression tokenization je postup využívající regulárních výrazů. Výhodou tohoto postupu, je možnost definovat pravidla pomocí regulárních výrazů, která upřesní dělení jednotlivých tokenů. Díky tomu, se dá zabránit chybám kterých se dopouští white space tokenization. [10]

3.5.2 Lemmanizace

V kontextu zpracování přirozeného jazyka se normalizace zabývá převáděním jednotlivých tokenů do jejich základní formy. Tímto způsobem se sníží počet unikátních tokenů v textu a odstraní se přebytečné informace. Mezi základní metody normalizace patří stemming a lemming. Výsledek procesu je možné vidět ve sloupci Lemma v tabulce 1.

Stemming je poměrně jednoduchý proces založený na předem definovaných hierarchicky seřazených pravidlech. Tato pravidla mohou vypadat následovně:

- SSES $\rightarrow$ SS
- IES $\rightarrow$ I
- SS $\rightarrow$ SS
- S $\rightarrow$ (prázdné)

Při použití těchto pravidel například na slově *cars*, se nejprve rozhodne jaké pravidlo je možné použít. Po postupném iterování nad

pravidly nakonec dospějeme k pravidlu čtvrtému, protože toto pravidlo se zabývá tokeny končícími písmenem *s*. Tímto získáme z množného čísla *cars*, jednotné číslo *car*. Výhodou tohoto procesu je rychlost. Nevýhodou však může být chybovost, která vzniká při aplikaci nesprávných pravidel, nebo při aplikaci pravidel na token, který je již v základním tvaru.

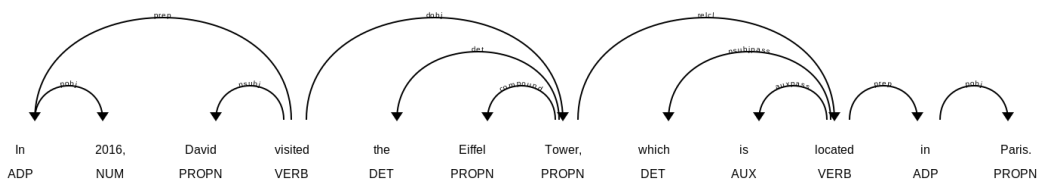
Lemming se oproti stemmingu kontroluje zdali se jednotlivé tokeny již vyskytují v poskytnutém korpusu. Pokud se daný token v korpusu vyskytuje, ponechá jeho aktuální podobu a pouze pokud tomu tak není dále hledá jeho základní formu. Díky tomuto postupu se zabrání většině chyb, kterých by se dopustil stemming. Tento proces je oproti stemmingu mnohem přesnější, ale zároveň náročnější. Důvodem je hledání tokenů v korpusu, který může být poměrně rozsáhlý. [10]

3.5.3 POS tagging

Pointou part of speech taggingu je přidělit jednotlivým tokenům značky, které vypovídají o jejich funkci ve větě. Část těchto tagů úzce spojena se slovními druhy a interpunkcí. Část z těchto tagů lze vidět ve sloupcích Pos a Tag v tabulce 1. Sloupec Pos je obecnější popis tokenu, který bývá nazýván coarse grained. Sloupec Tag je konkrétnější popis tokenu, také nazývaný fine grained.

Vybereme-li z tabulky 1 slova visited a located, vidíme, že obě slova mají ve sloupci Pos značku VERB (sloveso). Pokud se ovšem zaměříme na sloupec Tag, tak vidíme, že slovo visited má značku VBD (VBD: verb, past tense - sloveso, minulý čas) a slovo located má značku VBN (VBN: verb, past participle - sloveso, minulé příčestí). [10]

3.5.4 Dependency grammar



Obrázek 3: Dependency Grammar. Zdroj: [autor]

Dependency grammar slouží k rozpoznání vztahů mezi jednotlivými slovy ve větě. Příkladem je zpracovaná věta na obrázku 3. Cílem je najít kořen věty, od kterého se dále větví stromová struktura jednotlivých vztahů. Tyto vztahy se odvíjejí od Part of speech tagů zmíněných v předchozí části práce. Tato stromová struktura umožňuje části věty seskupit. Tyto skupiny slov budou následně dále využívány. [10]

3.5.5 Rozpoznávání pojmenovaných entit

Obrázek 4: Named entity. Zdroj: [autor]

Rozpoznání pojmenovaných entit je důležitou součástí získávání informací z přirozeného jazyka. Příklady názvů těchto entit jsou uvedeny ve sloupci Entity v tabulce 1, nebo na obrázku 4.

Na obrázku 4 je možno vidět, že skupina slov *the Eiffel Tower* získaná pomocí Dependency grammar, byla rozpoznána jako pojmenovaná entita LOC tj. lokace. [10]

3.6 Převod textu na řeč

Ve spojitosti s hlasovými asistenty je nutno zmínit i následný převod textu na hlas. Tento proces je nezbytnou součástí komunikace uživatele s hlasovým asistentem.

Obrázek 5: Blokové schéma převodníku textu na řeč. Zdroj: [11]

Systém pro převod textu na řeč se dá rozdělit na dvě hlavní části. První částí je předzpracování textu a druhou samotná syntetizace. Předzpracování textu probíhá v prvních třech blocích obrázku 5 a syntetizace v bloku posledním. [11]

3.6.1 Textová analýza

Prvním krokem této analýzy bývá nahrazení zkratek jejich plnou formou. Tento proces se řídí předdefinovanými pravidly. Tato pravidla určují které části textu budou nahrazeny za jiné.

Dalším krokem je normalizace. Tento proces se zabývá převodem čísel, nebo jiných znaků na jejich textovou podobu. Pokud například výslovnost řetězce *22* by měla být *dvacet dva*. Tento převod bývá založen na regulárních výrazech a měl by být aplikován rekurzivně, protože při samotném převodu mohou vzniknout znaky, které je nutno také převést. [11]

3.6.2 Fonetická analýza

Fonetická analýza se zabývá transkripcí textu. Tento proces se zabývá převodem textu na formu, kterou člověk slyší (popřípadě kterou vyslovuje). Tento převod úzce souvisí s problematikou popsanou v části Alofony, kdy slovo *ostrov* čteme *ostrof*. Samotný převod je uskutečněn pomocí rozsáhlých pravidel, která určují přepis jednotlivých fonémů a jejich spojení. A dále přepis fonémů do abecedy SAMPA (Speech Assessment Methods Phonetic Alphabet). [11]

3.6.3 Prozodická analýza

Prozódie popisuje zvukové vlastnosti jednotlivých fonémů. Bez těchto zvukových vlastností by převedená řeč zněla příliš monotónně. Hlavní činitelé změny vlastnosti jednotlivých fonémů jsou výška a hlasitost. Mezi prozodické jevy patří pauza, změna intonace, slovní a větný přízvuk a tempo řeči.

Výška základního tónu jednotlivých fonémů je určena podle typu věty tj. tázací, oznamovací atd. Každý typ věty má určenou startovací a koncovou výšku tónu. [11]

3.6.4 Syntéza

Základem hlasové syntézy je zvuková databáze. Pro přesné spojování řečových jednotek, musí mít zvuková databáze linearizovaný průběh základního tónu. Řečové jednotky jsou získávány z předmluvených textů, které jsou navrženy tak, aby obsáhly co největší počet jednotek. Kromě manuálního získávání jednotek, je též možné jednotky

segmentovat automaticky. K automatickému segmentování jednotek se používá například již zmíněné HMM, nebo DTW.

Následná syntetizace spočívá ve hledání jednotlivých řečových jednotek a jejich úpravě pomocí vlastností jednotlivých fonémů získaných pomocí Prozodické analýzy a takto upravené jednotky spojuje k sobě. [11]

4 Hlasové asistenty

Hlasové asistenty jsou uživatelským rozhraním, které je ovládáno přes hlasové příkazy. Hlasovým asistentem se rozumí software samotného asistenta. Tento software může být následně integrován do nejrůznějších zařízení. Tato část práce je zaměřena především na vybrané hlasové asistenty. Mezi tyto asistenty patří *Amazon Alexa* a *Google Assistant*. I přes tuto skutečnost principy zde popsané mohou být aplikovány i na řadu dalších hlasových asistentů.

4.1 Základní principy hlasových asistentů

Zpracování dotazovaného požadavku hlasovým asistentem je poměrně dlouhý proces, který se skládá z následujících částí:

- Zpracování požadavku
- Provedení akcí
- Vygenerování odpovědi

4.1.1 Spuštění interakce s hlasovým asistentem

Aby hlasový asistent poznal, kdy je na něj položen dotaz, musí neustále naslouchat zvukům okolního prostředí. Mezi těmito zvuky je neustále hledána spouštěcí fráze pomocí lokálního algoritmu pro rozpoznání hlasu. Tyto fráze mohou například vypadat následovně: *Alexa*, *Ok Google*. Pokud je tato spouštěcí fráze zaznamenána, hlasový asistent přejde do stavu interakce.

4.1.2 Interakce s hlasovým asistentem

Po spuštění samotné interakce hlasový asistent dále naslouchá a zpracovává požadavek uživatele. Zpracování samotného požadavku oproti

spouštění interakce již nemusí probíhat lokálně. Často je záznam požadavku streamován přímo na samotnou službu hlasového asistenta, kde je zpracován podrobněji.

4.1.3 Zpracování požadavku hlasovým asistentem

Po úspěšném převedení hlasu na text a následném podrobení textu metodám pro zpracování přirozeného jazyka, hlasový asistent identifikuje jakou akci má provést. Pokud tuto akci úspěšně nalezne, následně ji provede a podle výsledků vygeneruje odpověď. Pokud uživatel požaduje akci, která pro daného asistenta není definována, nebo ji nebylo možno rozpoznat, hlasový asistent se dotáže pro opakování požadavku.

4.1.4 Funkcionality hlasových asistentů

Aby hlasový asistent mohl interaktivně provádět příkazy uživatele, musí mít definované funkcionality. Tyto funkcionality se dají dělit na přímo vestavěné v samotném asistentovi, nebo na funkcionality třetích stran.

Pokud se jedná o funkcionality vestavěné, je na samotném asistentu aby tyto akce provedl a na základě jejich výsledku generoval odpověď.

Pokud se jedná o funkcionality třetích stran, hlasový asistent musí předat zpracované informace službě třetí strany, na které se samotná akce provede a následně bude vrácena vygenerovaná odpověď.

4.2 Zařízení hlasových asistentů

Aby mohl uživatel hlasové asistenty používat, musí být rozhraní samotných asistentů integrováno do uživatelských zařízení.

4.2.1 Dedikovaná zařízení hlasových asistentů

Jednou z nejčastějších fyzických forem hlasových asistentů jsou jejich dedikovaná zařízení. Nejjednodušší formou takového zařízení může být tzv. chytrý reproduktor (viz. obrázek 6). Interakce s takovým zařízením je založena pouze na hlasové formě. Výhodou těchto zařízení je poměrně malá pořizovací cena a naprosto dostačující funkcionalita.

Obrázek 6: Google Home Mini a Amazon Echo Dot 3. generace. Zdroj: [autor]

Další formou dedikovaného zařízení může být tzv. chytrý display (viz. obrázek 7). Oproti chytrým reproduktorům nabízí chytrý display možnost i vizuální a hmatové interakce. Tato vlastnost může být užitečná například při vyhledávání kuchařských receptů, nebo k přenášení obrazu kamery chytrého zvonku. [12]

Obrázek 7: Amazon Echo Show a Google Nest Hub. Zdroj: [12]

4.2.2 Chytré telefony s podporou hlasových asistentů

Většina dnes prodávaných chytrých telefonů umožňuje integrovat hlasové asistenty i přímo do jejich operačních systémů. Funkcionalita takových zařízení je srovnatelná s chytrými displeji, nicméně není potřeba pořizovat zvlášť dedikované zařízení, což se stává výhodou. Další výhodou může být fakt, že hlasový asistent v chytrém telefonu může být oproti dedikovanému zařízení mobilní.

4.2.3 Automobily s podporou hlasových asistentů

Ve snaze zabránit automobilovým nehodám způsobeným nepozorností řidiče, která je zapříčiněna interakcí s ovládacími prvky automobilu, se hlasové asistenty často stávají součástí softwaru samotného automobilu. Díky tomu řidič nemusí odvracet pohled od vozovky a může se plně věnovat řízení, čímž se zmenšuje reakční doba v případné rizikové situaci.

4.2.4 Ostatní zařízení s podporou hlasových asistentů

Kromě již zmíněných zařízení s podporou hlasových asistentů se na trhu objevují například i chytré hodinky, nebo chytré televize. Dále se na trhu objevuje i spousta dalších, toto je pouze základní výčet. Provozovatelé hlasových asistentů obvykle umožňují výrobcům získat pro svá zařízení certifikaci, která jim umožní integrovat hlasového asistenta do těchto zařízení.

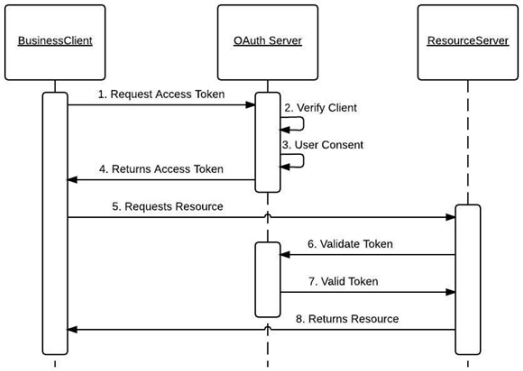
4.3 Analýza API vybraných hlasových asistentů

Pro rozšíření funkcí hlasové asistenty disponují API, která umožňují vytváření služeb třetích stran. Obecně by se tato API dala rozdělit do dvou skupin:

- Konverzační API [1, 2]
- API pro zařízení chytrých domácností [3, 4]

4.3.1 Autentizace služeb třetích stran

Pro autentizaci a identifikaci uživatelů vybrané hlasové asistenty používají OAuth2. Tato technologie slouží k autentizaci uživatelů a je provozována výhradně pomocí HTTPS (Hypertext Transfer Protocol Secure) protokolu.

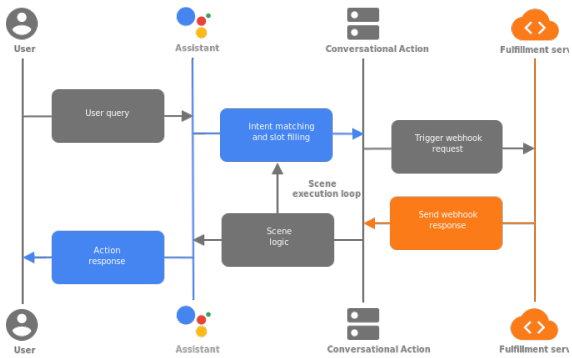


Obrázek 8: OAuth2. Zdroj: [13]

V praxi je tato technologie využívána tak, že služba třetí strany má jednotlivé asistenty zaregistrovány jako aplikace, kterým může o svých uživatelích poskytnout informace. Tento proces spočívá v tom, že jednotlivým asistentům poskytne tzv. *client token*, *client*

secret a popřípadě tzv. *scopes*. Pomocí těchto údajů se aplikace hlasového asistenta identifikuje a požádá uživatele o svolení k přístupu k jeho datům. Pokud uživatel k přístupu svolí, aplikaci hlasového asistenta je přidělen tzv. *acess token*. Následně se služba hlasového asistenta může dotazovat informace o uživateli služby třetí strany. Tyto informace jsou limitovány pouze na informace, které je možné získat v přidělených scopes. [14, 15]

4.3.2 Volání služeb třetích stran



Obrázek 9: Komunikace mezi službami hlasových asistentů. Zdroj: [16]

Oba z vybraných hlasových asistentů používají ke komunikaci se službou třetí strany technologii Webhook. Tato technologie je založena na jednoduchém předávání serializovaných dat pomocí HTTPS POST požadavků. U této technologie není striktně určena struktura předávaných dat. Co se týče serializace a deserializace samotných dat, nejpoužívanějším formátem se stává JSON (JavaScript Object Notation). Na obrázku 9 je možné tento proces pozorovat mezi prvky *Conversational Action* a *Fulfillment server*.

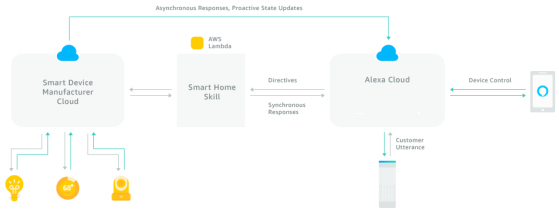
Pro konverzační rozhraní se tyto zprávy skládají výhradně z názvu identifikované akce a jejích parametrů. Pro rozhraní chytré domácnosti lze zprávy rozdělit do čtyř hlavních skupin.

První skupinou je synchronizace zařízení uživatele. Tato synchronizace slouží k ohlášení, jaká chytrá zařízení uživatel vlastní a jaké ovládací prvky tato zařízení mají. Společnost Amazon tyto zprávy nazývá *Discovery* a společnost Google *SYNC*.

Druhou skupinu tvoří zprávy sloužící k oznámení stavů ovládacích prvků jednotlivých zařízení. Amazon tuto skupinu zpráv nazývá *ReportState* a Google *QUERY*.

Třetí skupina slouží k nastavení stavů ovládacích prvků na jednotlivých zařízeních. Amazon tyto zprávy nazývá specificky pro každý druh ovládacího prvku zvlášť. Příkladem může být *AdjustPercentage*. Google tuto skupinu zpráv nazývá *EXECUTE*.

Poslední skupina slouží k oznámení o zrušení autorizace pro danou aplikaci třetí strany. Společnost Amazon tuto skupinu zpráv nepoužívá. O zrušení autorizace se aplikace třetí strany dozví až v případě, že hlasový asistent požádá znovu o novou autorizaci. Google naopak dbá na zrušení nevyužívaných access tokenů a v případě zrušení autorizace odešle zprávu, kterou nazývá *DISCONNECT*.



Obrázek 10: Amazon smart home komunikace. Zdroj: [3]

Společnost Amazon pro aplikace třetích stran zabývající se chytrými domácnostmi, vyžaduje navíc definici AWS Lambda funkce (viz. obrázek 10). Tato funkce slouží jako prostředník mezi službou asistenta a aplikací třetí strany. Toto řešení umožňuje definici i jiného způsobu komunikace, ovšem obvyklou strategií bývá pouhé předání informace službě aplikaci strany.

Pokud se jedná o komunikaci od aplikace třetí strany k hlasovému asistentovi, která je používána výhradně s aplikacemi zaměřenými na chytrou domácnost, je používáno klasické REST API a společnost Google pro svého hlasového asistenta umožňuje i alternativu pomocí gRPC protokolu. Tento způsob komunikace je využíván při ohlašování změn stavu jednotlivých zařízení. [3, 17]

4.3.3 Konverzační funkcionality

Konverzační funkcionality třetích stran umožňují definovat vlastní konverzační rozhraní. Pro tuto definici má každý z vybraných hlasových asistentů nástroj, pomocí kterého je možné definovat jednotlivé akce a jejich spouštěcí fráze. Ve spouštěcích frázích lze dále definovat i parametry jednotlivých akcí. Tyto parametry mohou mít specifické typy. Příklady jednotlivých typů jsou v tabulce 2.

Tabulka 2: Příklad definovaných typů pro konverzační akce hlasových asistentů. Zdroj: [autor]

Název	Amazon	Google
Datum	DATE	Date
Číslo	NUMBER	Number
Cokoli	SearchQuery	Any

Při zpracovávání příkazu uživatele hlasový asistent tyto parametry vyhledá pomocí technik pro zpracování přirozeného jazyka. Následně jsou tyto parametry spolu s rozpoznanou akcí předány službě třetí strany, která tyto parametry využije při vykonávání akcí a tvorbě odpovědi. Pokud se při rozpoznání parametrů vyskytne chyba, je možné definovat i fráze pro opětovné dotázání na specifický parametr.

Akce samotné je možné i řetězit. Tato vlastnost je využívána k vytvoření kontextu o uživatelově požadavku. Příkladem může být akce, která slouží k objednání letenek. Po požadavku na zakoupení letenky, se hlasový asistent uživatele postupně dotazuje na místo odletu, místo příletu a čas odletu. Díky tomu si vytvoří kontext, na základě kterého je schopný vyhledat nejvhodnější let a provést zakoupení tohoto letu. [1, 2]

4.3.4 Funkcionality chytrých zařízení

Oproti konverzačním funkcionalitám, funkcionality chytrých zařízení neumožňují definici vlastního konverzačního rozhraní. Rozhraní těchto funkcionalit je tvořeno automaticky na základě ovládacích prvků jednotlivých zařízení. Tato zařízení lze dále zařadit do skupin, které mohou reprezentovat celou domácnost, nebo například jen místnost domácnosti. [1, 4]

Příkladem může být chytrá lampa, kterou je možné rozsvítit, zhasnout a nastavit intenzitu světla. Tudíž konverzační rozhraní pro tuto lampu bude obsahovat akci pouze pro spuštění, vypnutí a nastavení intenzity světla. Dále tato lampa může být zařazená do skupiny *living room*, která zahrnuje obývací pokoj domácnosti. Tato lampa může být potom ovládána například následujícími příkazy:

- Turn on the lamp.
- Turn of the lamp.
- Set light brightness to 80 percent.
- Incerease light brightness.
- Turn on the living room lights.
- Is living room light on?

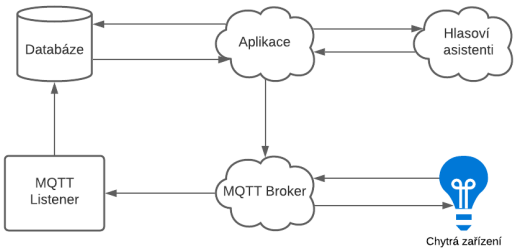
4.3.5 Ostatní funkcionality

Kromě konverzačních funkcionalit a funkcionalit chytrých domácností oba z vybraných hlasových asistentů umožňují v jejich vývojovém prostředí zvolit i jiné typy aplikací třetích stran. Mezi tyto typy patří například vzdělávání, hry atd. Tyto typy jsou pouze upřesněním, o jakou aplikaci třetí strany se jedná, nicméně výstavba samotné aplikace probíhá pomocí dvou již zmíněných API. [1, 2]

5 Návrh aplikace

Kapitola popisuje návrh aplikace, která slouží k propojení vybraných hlasových asistentů s prvky chytré domácnosti. Její součástí je i výběr vhodných technologií a vhodného prostředí pro nasazení této aplikace.

5.1 Koncept



Obrázek 11: Blokový diagram částí aplikace. Zdroj: [autor]

Základní koncept aplikace se skládá z několika hlavních prvků. Prvním je databáze, ve které jsou uloženy veškeré informace o uživatelích, jejich chytrých zařízeních a autentizační informace poskytované hlasovým asistentům. Druhou částí je samotná aplikace, jejíž úkolem je poskytovat informace o chytrých zařízeních hlasovým asistentům a samotná správa chytrých zařízení. Další částí je MQTT broker, který zajišťuje komunikaci mezi chytrými zařízeními a aplikací. MQTT listener se stará o příjem aktuálních informací od MQTT brokeru z chytrých zařízení a o promítnutí těchto informací do databáze aplikace.

5.1.1 Koncept aplikace

Vzhledem k autentizačním a komunikačním metodám hlasových asistentů popsaných v předešlé části práce je aplikace navržena jako webová aplikace. Kromě registrace a přihlášení uživatelů, umožňuje uživatelům definovat svá vlastní chytrá zařízení. Díky tomu spadá do využití aplikace i vývoj chytrých zařízení. Samotné zdrojové kódy aplikace jsou volně dostupné a je možná jejich úprava pro vlastní využití tzn. aplikace je open source. Případným využitím upravené aplikace může být služba výrobce chytrých zařízení.

Hlavním úkolem aplikace je zprostředkovat komunikaci se službami vybraných hlasových asistentů. Dalším úkolem aplikace je přenášení dat mezi chytrými zařízeními a aplikací samotnou.

5.1.2 Koncept MQTT listeneru

MQTT listener je vybudován jako neoddělitelný prvek aplikace. Pro testovací účely je možné jeho spuštění se samotnou aplikací (tzn. ve vlastním vlákně) a pro účely rozkládání zátěže na servery aplikace i samostatně (tzn. samostatný proces). Tato funkcionalita je nastavitelná v konfiguračním souboru aplikace.

Jeho hlavním úkolem je z databáze zjistit která MQTT topics odebírat, přihlásit jejich odběr a při příchodu nové informace tuto informaci promítnout do databáze. V případě odstranění zařízení naopak zrušení odběru jeho MQTT topics.

5.2 Výběr technologií

Výběr technologií je nedílnou součástí návrhu aplikace. Technologie musí být vybrané s ohledem na koncept aplikace a tak, aby zajistily co největší kompatibilitu mezi jejími částmi a tím usnadnily vývoj samotné aplikace.

5.2.1 Běhové prostředí

Pro běhové prostředí aplikace byl vybrán programovací jazyk Python. Tento programovací jazyk je high-level, objektově orientovaný a interpretovaný jazyk, který umožňuje dynamické typování. Díky tomu, že je tento programovací jazyk open source, disponuje velikou škálou distribucí a podporou různých platforem. Dále také disponuje jednoduchou a snadno čitelnou syntaxí. Vzhledem k těmto vlastnostem, umožňuje rychlý vývoj aplikací. [18]

5.2.2 Knihovny

Jako webový framework byl vybrán Flask. Tento minimalistický framework je velmi tvárný, díky tomu je jeho využití velice široké a skvěle zapadá do základního konceptu aplikace. Flask je vytvořen pomocí knihovny Werkzeug. Díky tomu obsahuje vývojový WSGI (Web Server Gateway Interface) a debugger, který usnadňuje práci při vývoji

aplikace. Součástí Flasku je i šablonovací stroj Jinja2, který zajišťuje jednoduché vytváření uživatelských rozhraní pomocí vyplňování HTML (Hypertext Markup Language) šablon. [19]

Pro produkční WSGI je vybrán Gevent WSGI. [20] Nicméně WSGI je možné měnit v závislosti na nasazení aplikace a dalším vhodným kandidátem může být například Gunicorn. [19]

Další výhodou Flasku je podpora nejrůznějších knihoven. Mezi knihovny požité při tvorbě aplikace patří Flask-WTF, která usnadňuje zjednodušené vytváření formulářů a jejich následnou validaci. [21] Dále například Flask-Security, tato knihovna zajišťuje přihlášení a registraci uživatelů a zároveň i poskytuje možnost RBAC (Role Based Access Control). Mezi její další přednosti patří i možnost automatického zasílání informačních emailů uživateli o změnách hesla atd. [22] Pro vytvoření OAuth2 rozhraní je použita knihovna *authlib*. Pro snadnou tvorbu REST API rozhraní, je použita knihovna Flask-Restx. Výhodou této knihovny je i automatické generování dokumentace API pomocí nástroje Swagger. [23]

5.2.3 Databázové řešení

Databázové řešení je založeno na ORM (Object Relational Mapping), které usnadňuje práci s jednotlivými entitami mapováním na třídy objektů, definující tabulky databáze. Pro toto řešení je vybrána knihovna Pony. Tato knihovna podporuje řadu SQL (Structured Query Language) dialektů, díky čemuž je vhodná pro práci s širokou škálou relačních databázových strojů. Další výhodou je možnost psaní dotazů jak pomocí SQL, tak i pomocí Python lambda výrazů, které jsou následně přeloženy do daného SQL dialektu. Pony disponuje i vestavěnou podporou pro již zmíněný Flask. Nevýhodou Pony, je nedostačující podpora automatických migrací dat při změnách definic tabulek. [24]

5.2.4 MQTT Broker

Pro testovací účely aplikace využívá veřejně dostupný MQTT broker HiveMQ. Pro produkční nasazení je vhodné využívat soukromý broker. Důvodem je bezpečnost dat a lepší možnost autentizace. [25]

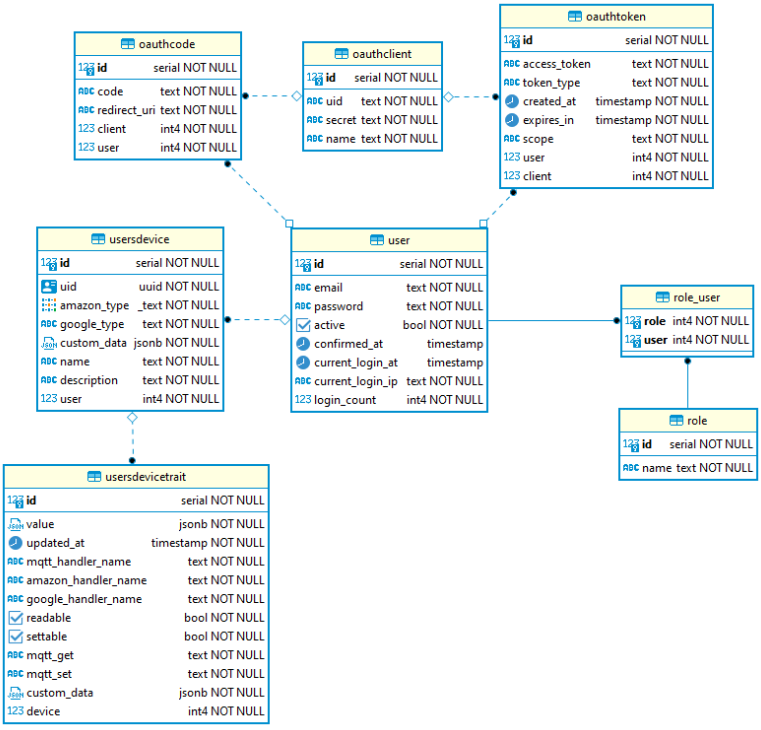
5.2.5 Hosting

Vzhledem k požadavkům na veřejně dostupnou adresu a HTTPS protokol, je nutné aplikaci testovat a provozovat na serveru, který tyto

požadavky splňuje. Z tohoto důvodu je vybrána služba Heroku. Tato služba umožňuje hostovat aplikace psané v široké škále jazyků, mezi které patří i vybraný Python. Kromě toho tato služba poskytuje i PostgreSQL databázi, která je využívána na samotném hostingu. Další výhodou Heroku, je možnost nahrávání zdrojových kódů a automatické vytváření virtuálního prostředí pomocí nástroje Heroku CLI. [26] Samotné nahrávání kódu probíhá pomocí nástroje Git.

5.3 Datové modely aplikace

Jak již bylo zmíněno, aplikace používá ORM, které umožňuje definici databázových tabulek na základě modelových tříd.



Obrázek 12: ER Diagram datových modelů aplikace. Zdroj: [autor]

5.3.1 Datový model uživatele

Datový model uživatele (tabulka *user*) obsahuje běžné atributy, mezi které například patří email, heslo, aktivní status, nebo čas posledního přihlášení. Kromě těchto klasických atributů, má dále také vazbu na uživatelské role (tabulky *role_user* a *role*). Tato vazba je klíčová pro RBAC. Další vazby dále vedou na modely OAuth a model samotných chytrých zařízení (viz. obrázek 12).

5.3.2 Datové modely OAuth

Hlavním z OAuth modelů je *oauthclient*. V tomto modelu jsou definovány identifikační údaje poskytnuté službám jednotlivých hlasových asistentů. Na základě těchto údajů může aplikace službám hlasových asistentů poskytnout auth code a po schválení uživatelem následně i access token.

5.3.3 Datové modely chytrých zařízení

Samotné chytré zařízení je definován jako *usersdevice*.

Atribut *uid* slouží k identifikaci chytrých zařízení hlasovým asistentům. Dle dokumentace hlasových asistentů, musí být tento identifikátor unikátní v globálním měřítku. Pro zaručení unikátnosti tohoto atributu je použito *uuid4*. Dalším atributem je *amazon_type*. Dle dokumentace hlasového asistenta Amazon Alexa, jednotlivá chytrá zařízení mohou spadat do několika předdefinovaných kategorií. Z toho důvodu tento atribut může obsahovat seznam znakových řetězců, které tyto kategorie popisují. Atribut *google_type* popisuje typ zařízení pro Google Assistant. Narozdíl od Amazon Alexa, chytré zařízení může mít pouze jeden typ. Pokud se zařízení dá shrnout do více kategorií, je možné toto zařízení virtuálně rozdělit na několik zařízení. *Custom_data* je JSON atribut obsahující doplňující informace o chytrém zařízení, které umožňují flexibilitu při jejich definici. Zbylé atributy *name* a *description* slouží k určení názvu zařízení a jeho popisu.

Jednotlivé ovládací prvky chytrého zařízení jsou definovány jako *usersdevicetrait*. Hlavním atributem je samotná hodnota ovládacího prvku. Z důvodu různorodosti dat, je definována jako JSON. Atribut *updated_at* uchovává čas poslední změny hodnoty ovládacího prvku. Na základě tohoto atributu je možné zjistit, zdali je chytré zařízení aktivní (online). *Mqtt_handler_name*, *amazon_handler_name* a *google_handler_name* uchovávají názvy handlerů jednotlivých ovládacích prvků. *Readable* a *settable* souží k definici čtení a zápisu hodnot ovládacích prvků. *Mqtt_get* a *mqtt_set* uchovávají informace o MQTT tématech sloužících pro přenášení hodnoty ovládacích prvků z a na chytrá zařízení. Posledním atributem je podobně jako u samotných zařízení *custom_data*, který umožňuje flexibilitu definice jednotlivých ovládacích prvků a je též definován jako JSON.

5.4 Návrh obsluhy ovládacích prvků chytrých zařízení

Vzhledem k různorodosti dat ovládacích prvků chytrých zařízení, je nutné s každým druhem dat a ovládacím prvkem zacházet rozdílně. Pro zajištění kompatibility dat, je nutné definovat, jak zacházet s daty pro MQTT, Amazon Alexa a Google Assistant zvlášť.

5.4.1 Zpracování dat a požadavků

Zpracování jednotlivých dat ovládacích prvků je založeno na třídách definujících jejich obsluhu a konverzi. Tyto třídy budou dále nazývány jako trait handlery. Tyto handlery jsou rozděleny do tří skupin:

1. MQTT handlery
2. Amazon handlery
3. Google handlery

Hlavním úkolem první skupiny je přijímat a odesílat data v určeném formátu přes MQTT. Úkolem skupiny druhé je zajistit správný formát dat pro hlasového asistenta Amazon Alexa a obsluhu případných požadavků. Kromě toho i definovat kompatibilitu s hanlery z první skupiny. Skupina třetí plní podobný úkol jako skupina druhá, ovšem pro hlasového asistenta Google Assistant. Každý z handlerů má staticky definované jméno, které umožní jeho identifikaci a popis, který bude následně využit při tvoření nápovědy, nebo dokumentace. Výhodou použití tříd, je možnost využití dědičnosti, díky které je definice jednotlivých handlerů výrazně zjednodušena.

5.4.2 Práce s handlery ovládacích prvků

Kromě handlerů samotných, je nutné vytvořit jejich správce. Tento správce se nazývá trait handler controller. Tento controller má definovaný seznam handlerů z každé ze tří již míněných skupin a umožňuje jejich vyhledávání a zjišťování kompatibility mezi skupinami.

6 Implementace aplikace

Hlavní strukturu aplikace lze rozdělit dle následného seznamu:

- /application/**/*.py - zdrojové kódy aplikace,
- /static/**/* - statické soubory,
- /templates/**/*.html - HTML šablony uživatelského rozhraní,
- /serve.py - soubor pro spuštění aplikace,
- /config.py - soubor nastavení aplikace,
- /listener.py - soubor pro samostatné spuštění listeneru,
- /requirements.txt - soubor obsahující seznam využívaných knihoven.

6.1 Implementace uživatelského rozhraní

```
58  @app.route('/dashboard')
59  @login_required
60  def dashboard():
61      devices = current_user.devices.order_by(UsersDevice.name)
62      return render_template(
63          'dashboard.html',
64          devices=devices,
65          online_devices=[
66              device
67              for device in devices
68              if device.online
69          ],
70          mqtt_host=MQTT_HOST,
71          mqtt_port=MQTT_PORT,
72          trait_controller=trait_controller
73      )
```

Obrázek 13: Ukázka view funkce uživatelského rozhraní. Zdroj: [autor]

Uživatelské rozhraní je tvořeno podle návrhového vzoru MVT (Model View Template). Modelem v tomto případě jsou samotné ORM modely popsané v návrhu aplikace. Views jsou definovány jako jednoduché funkce, jejichž cílem je vracet vyplněnou HTML šablonu. Příklad takové funkce je na obrázku 13. Z obrázku je též možné vidět tzv. dekorátory, které v tomto případě určují URL na které bude tato funkce dostupná (@*app.route*(...)) a že tato funkce vyžaduje přihlášení uživatele (@*login_required*). V těle této funkce je možné vidět i samotnou práci s ORM modely (*devices* = *current_user.device.order_by*(...)).

Následná návratová hodnota je vytvořena pomocí funkce *render_template(...)*, která je součástí již zmíněného šablonovacího stroje Jinja2 a generuje vyplněnou HTML šablonu.

6.1.1 Běžné uživatelské rozhraní

Běžné uživatelské rozhraní je implementováno podobným způsobem, jako funkce na obrázku 13, která je též jeho součástí. Knihovna *Flask-Security* dále definuje views starající se o správu uživatelů. Příkladem takových views může být */login*, */register* atd. Kromě views definovaných knihovnou *Flask-Security*, jsou definovány i views v následujícím seznamu:

- / - Domovská stránka.
- /dashboard - Domovská stránka pro přihlášené uživatele.
- /tos a /privacy - Podmínky služby a zásady soukromí.
- /api - Dokumentace REST API.
- /handlers a /handler/<handler_name> - Dokumentace handlerů zmíněných v návrhu aplikace.

6.1.2 CRUD rozhraní pro práci s chytrými zařízeními

Jak už sanotný význam zkratky CRUD (Create Read Update Delete) napovídá, je nutné vytvořit view funkce obstarávající tyto základní funkcionality pro samotná chytrá zařízení a i pro jejich ovládací prvky. Tyto views se nijak významně neliší od běžných views, ale jejich hlavní odlišností je využití definic formulářů a jejich validace pomocí knihovny *Flask-Wtf*.

```
129 class DeviceAddForm(Form):
130     name = StringField('Device name', validators=[DataRequired()])
131     description = StringField('Device description', validators=[DataRequired()])
132     amazon_type = StringListField('Amazon type')
133     google_type = StringStripField('Google type')
134     custom_data = JSONDictField('JSON formatted device custom data (Add or edit only if you know what you are doing.)')
135     submit = SubmitField('Add')
136
137
138 class DeviceEditForm(DeviceAddForm):
139     submit = SubmitField('Edit')
```

Obrázek 14: Ukázka definice formuláře uživatelského rozhraní. Zdroj: [autor]

Tyto formuláře jsou definovány jako třídy se statickými atributy, které definují jednotlivá pole formuláře a jejich datové typy a případné validátory (viz. obrázek 14). Definice těchto formulářů se odvíjí od samotných ORM modelů. CRUD rozhraní je definováno podle následujícího seznamu:

- /device/<int:device_id> - zobrazení specifického chytrého zařízení,
- /device/add - přidání chytrého zařízení,
- /device/edit/<int:device_id> - editace chytrého zařízení,
- /device/delete/<int:device_id> - odstranění chytrého zařízení,
- /device/<int:device_id>/trait/add - přidání ovládacího prvku chytrého zařízení,
- /device/<int:device_id>/trait/edit/<int:trait_id> - editace ovládacího prvku chytrého zařízení,
- /device/<int:device_id>/trait/delete/<int:trait_id> - odstranění ovládacího prvku chytrého zařízení.

6.2 Implementace OAuth rozhraní

Samotné OAuth2 rozhraní je tvořeno na základě knihovny *authlib*. Toto rozhraní je podobně jako uživatelské rozhraní tvořeno pomocí view funkcí. Ovšem kromě HTML dokumentu, který se dotazuje uživatele zdali chce určitému hlasovému asistentovi o sobě poskytnou data, tak může vracet i tokeny potřebné k autentizaci pomocí OAuth2.

OAuth2 rozhraní je definováno na /oauth/authorize a /oauth/token. Tyto URL je nutné nastavit jako výchozí body autentizace ve vývojových nástrojích jednotlivých hlasových asistentů.

6.3 Implementace webhooku

Vzhledem k tomu, že webhook slouží pro komunikaci se službami hlasových asistentů, je podstatnou částí aplikace. Obsluha jednotlivých požadavků je rozdělena podle vybraných hlasových asistentů.

6.3.1 Webhook rozhraní

```python
@app.route('/fulfillment/<assistant>', methods=['POST'])
@oauth_required()
def fulfillment(assistant: str):
    request_dict = request.get_json()
    app.logger.debug(json.dumps(request_dict))
    if assistant == 'amazon':
        response_dict = amazon.handle(request_dict)
    elif assistant == 'google':
        response_dict = google.handle(request_dict)
    else:
        return abort(404)
    app.logger.debug(json.dumps(response_dict))
    return response_dict
```

Obrázek 15: Definice webhook rozhraní. Zdroj: [autor]

Webhook rozhraní je definováno na /fulfillment/<assistant>, kde za proměnou *assistant* lze doplnit *amazon* pro hlasového asistenta Amazon Alexa a *google* pro hlasového asistenta Google Assistant. Z těla požadavku je vždy získán JSON, který je deserializován a následně zpracován pomocí funkcí *amazon.handle(...)* a *google.handle(...)* dle proměnné *assistant*. Toto rozhraní je zabezpečeno pomocí dekorátoru *@oauth_required* který zaručí, že data budou přijímána a poskytována jen ověřeným službám tzn. službám hlasových asistentů (viz. obrázek 15).

6.3.2 Obsluha požadavků na webhook

```python
def handle_discovery() -> dict:...

def handle_report(request_dict: dict) -> dict:...

def handle_command(request_dict: dict) -> dict:...

def handle(request_dict: dict) -> dict:...
```

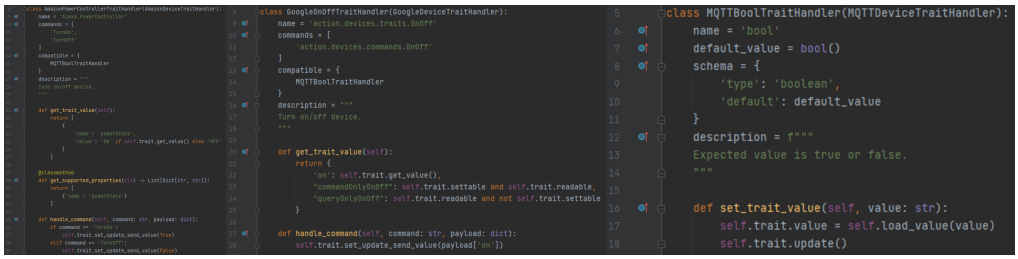
```python
def handle_sync(request_dict: dict) -> dict:...

def handle_query(request_dict: dict) -> dict:...

def handle_execute(request_dict: dict) -> dict:...

def handle(request_dict: dict) -> dict:...
```

Obrázek 16: Rozdělení obsluhy požadavků na webhook. Zdroj: [autor]

Jak již bylo zmíněno, obsluhu webhooku je nutné rozdělit do dvou částí podle toho o jakého asistenta se jedná. A následně rozdělit podle druhu požadavku, tak jak je to popsáno v sekci Volání služeb třetích stran. Pro oba asistenty je požadavek nejprve předán funkci *handle(...)*, která určí typ požadavku a následně požadavek předá jedné

z funkcí *handle_...(...)* (viz. obrázek 16). Odpověď požadavku je generována na základě ORM tříd chytrých zařízení a jejich ovládacích prvků, za pomoci trait handlerů popsaných v návrhu aplikace.

6.3.3 Implementace trait handlerů



Obrázek 17: Ukázka definice trait handlerů. Zdroj: [autor]

Jak již bylo zmíněno v návrhu aplikace, definice trait handlerů je založena na dědičnosti tříd. Na obrázku 17 je možné vidět definici handlerů pro Amazon Alexa, Google Assistant a následně i pro data přicházející z MQTT brokeru.

Klíčovými funkcemi handlerů hlasových asistentů jsou *get_trait_value(...)*, jejichž úkolem je převádět uložené hodnoty ORM modelů z databáze aplikace. Jak je možné vidět z definice těchto funkcí, každý hlasový asistent vyžaduje jiný formát dat. Další klíčovou funkcí je *handle_command(...)*, která se stará o provádění příkazů od hlasového asistenta a promítnutí dat pomocí ORM modelu zpět do databáze.

Pro MQTT handlery je klíčový atribut *schema*. Jedná se o atribut definující JSON schéma, které je používáno pro validaci dat, jež MQTT listener zaznamenává zpět do databáze. O samotné zapsání dat do databáze se stará funkce *set_trait_value(...)*, která převede data do unifikovaného formátu a následně je zaznamená.

6.4 Implementace MQTT listeneru

Listener je definován tak, že může běžet vně aplikace, ale zároveň i samostatně. Pro spuštění vně aplikace, je nutné vytvořit vlastní vlákno, na kterém bude opakovaně prováděna funkce *loop(...)*, znázorněná na obrázku 18. O to se stará privátní funkce *_worker(...)*. O spuštění vlákna a případné zastavení se starají funkce *start(...)* a *stop(...)*. Pokud je listener spuštěn samostatně, je naopak využita funkce *loop_forever(...)*. Statická funkce *on_message(...)* se stará o obsluhu přijatých dat z MQTT listeneru.

Obrázek 18: Definice MQTT listeneru. Zdroj: [autor]

6.5 Implementace REST API

Implementace REST API je založena na knihovně Flask-Restx, která umožňuje definici jednotlivých koncových bodů formou tříd. Tyto třídy mají následně definované funkce *get(...)*, *post(...)*, *put(...)* a *delete(...)*, které odpovídají HTTP metodám použitým k volání jednotlivých koncových bodů. Pro zabezpečení těchto funkcí je možné opět použít dekorátor *@login_required*. Tento dekorátor umí kromě práce s tokeny uloženými v Cookies požadavku i pracovat s tokeny přiloženými v hlavičce požadavku. Běžně používaný název tokenu v hlavičce požadavku je *Authentication-Token*.

Komunikace s koncovými body REST API probíhá výhradně pomocí JSON dat, která jsou předávána v těle požadavku. Stejně tak jako u CRUD rozhraní, je nutné tato data validovat. K tomu knihovna *Flask-Restx* používá tzv. *RequestParser*, jehož úkolem je JSON data z těla požadavku deserializovat a následně pomocí JSON schémat definovaných pro každý vstupní model i validovat. Tato vstupní a výstupní schémata jsou následně použita i při generování Swagger dokumentace celého REST API. Celé REST API je dostupné na koncových bodech:

- /api/v1/device/<str:uid> - koncový bod REST API pro chytrá zařízení.
- /api/v1/trait/<str:uid> - koncový bod pro ovládací prvky chytrých zařízení.

7 Testování aplikace

Pro ověření správné funkce aplikace je nutné aplikaci testovat. Testování je nedílnou součástí vývojového procesu, nicméně pro odhalení některých chyb a nedostatků (způsobených převážně v běhovém čase), je nutné tyto testy rozdělit a provádět dlouhodobě.

7.1 Testování ze strany hlasových asistentů

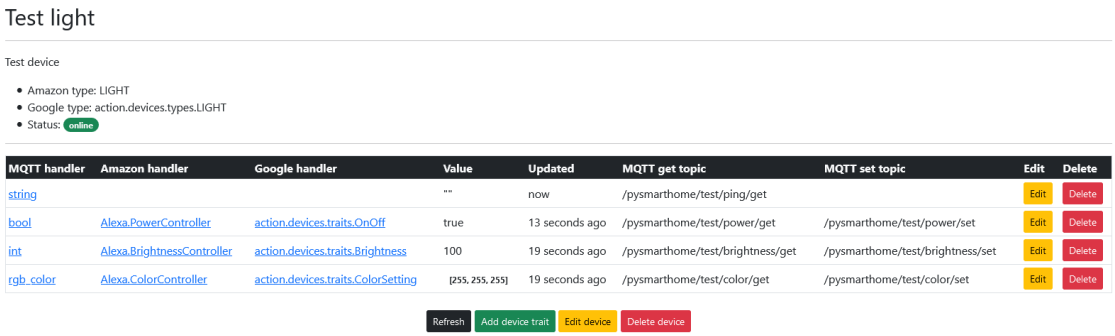
Pro testování aplikací třetích stran má vývojové prostředí vybraných hlasových asistentů jednoduchý nástroj, jehož úkolem je simulovat hlasového asistenta. U hlasového asistenta Amazon Alexa, je tento nástroj specificky rozdělen pro druh aplikace třetích stran. Pro chytré domácnosti tento nástroj umožňuje nahlížet do zasílaných zpráv mezi službou třetí strany a službou hlasového asistenta. Tato funkcionalita umožňuje odhalit chybný formát JSON struktury zprávy. Dále také umožňuje vybrat specifické zařízení, jeho ovládací prvky a následně pro něj spustit automatické testování. Při samotném testování jsou zkoušeny funkce vybraných ovládacích prvků a výsledek (JSON popisující test) je možné analyzovat v samotném vývojovém prostředí, nebo jej stáhnout a provést podrobnější analýzu. [27]

Hlasový asistent Google Assistant má ve svém vývojovém prostředí simulátor, který se poměrně liší od testovacího nástroje asistenta Amazon Alexa. Ačkoli je možné podobným způsobem testovat vytvářenou aplikaci třetí strany, tento nástroj poskytuje pouze možnost interakce formou stejnou jako s reálným hlasovým asistentem. Pro testování aplikací třetích stran zaměřených na chytré domácnosti, Google Assistant poskytuje oddělený nástroj, který je srovnatelný s nástrojem Amazon Alexy, ovšem nelze testovat specifická zařízení, ale pouze všechna zařízení. [28]

7.2 Testování virtuálních zařízení

Při testování virtuálních zařízení je nutné ho definovat v aplikaci. Pro testovací účely byla definována chytrá lampa, mezi jejíž ovládací prvky patří funkce On/Off, nastavení barvy světla a nastavení intenzity světla (viz. obrázek 19). Toto zařízení spadá do skupiny *LIGHT* pro asistenta Amazon Alexa a do skupiny *action.devices.types.LIGHT* pro asistenta Google Assistant. V definici ovládacích prvků je možné vidět prvek, který nemá definován žádná typ pro hlasové asistenty nicméně

má definovyný MQTT topic na */pysmarthome/test/ping/get*. Tento prvek je určen k zjišťování statusu zařízení tzn. online/offline. Zbytek ovládacích prvků je definován dle předpokládaných hodnot.

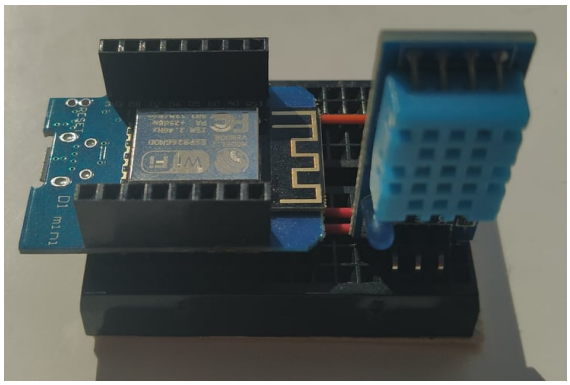


Obrázek 19: Definice virtuální lampy v aplikaci. Zdroj: [autor]

Po definici zařízení v aplikaci, bylo zařízení vytvořeno ve formě jednoduchého programu v příkazovém řádku. Toto zařízení je součástí zdrojových kódů aplikace a nachází se v kořenovém adresáři */test_light.py*. Při testování tohoto virtuálního zařízení bylo zjištěno, že aplikace funguje podle očekávání. Přenos dat a reakce na změnu dat je díky MQTT protokolu téměř instantní.

7.3 Testování s reálným zařízením

Pro testování s reálným zařízením byla vybrána vývojová deska LOLIN D1 mini. Tato deska je založena na mikrokontroléru ESP-8266EX, který podporuje WiFi (Wireless Fidelity) konektivitu. [29] K vývojové desce byl připojen senzor teploty vzduchu a relativní vlhkosti vzduchu, jehož hodnoty jsou odesílány na MQTT broker aplikace (viz. obrázek 20).



Obrázek 20: Reálné chytré zařízení. Zdroj: [autor]

V aplikaci bylo definováno toto zařízení jako *TEMPERATURE_SENSOR* pro asistenta Amazon Alexa a jako *action.devices.types.SENSOR* pro asistenta Google Assistant. U zařízení je nutné definovat ovládací prvek pro čtení teploty a vlhkosti. Vybrané hlasové asistenty podporují definici ovládacího prvku teploty, ovšem co se týče vlhkosti vzduchu, narážíme na problém u hlasového asistenta Amazon Alexa. API chytrých domácností tohoto asistenta tuto funkcionalitu bohužel nepodporuje a proto je definováno pouze pro asistenta Google Assistant. Po definici a testování tohoto zařízení, bylo zjištěno, že aplikace je funkční a dokáže pracovat i s reálnými zařízeními.

8 Závěr

Chytrá zařízení ve spojení s automatizačními aplikacemi a hlasovými asistenty jsou bezesporu přínosem do moderní domácnosti. Jejich funkcionality mohou výrazně ulehčit problémy každodenního života. Pomoci mohou také lidem s handicapem. Chytrých zařízení na trhu stále přibývá. Jejich využití je velmi široké a sahá od chytrých vypínačů a světel až po chytré termostaty a zabezpečení obydlí.

V teoretické části práce byl popsán princip hlasových asistentů. Je důležité si uvědomit, že proces zpracování vysloveného požadavku je dlouhý a obtížný. Samotný záznam hlasu a jeho následné převedení na text je velmi složitá problematika. Následné zpracování přirozeného jazyka má v informačních technologiích širokou škálu využití. Od vyhledávání podstatných informací z textu, QnA (Question and Answer) botů až po samotné hlasové asistenty. Převedení textu na hlas je též složitý proces, jehož využití je široké.

Z části práce zabývající se hlasovými asistenty je možné nahlédnou do pozadí procesů používaných vybranými hlasovými asistenty. API hlasových asistentů využívá webhook technologii, která nemusí být známá pro běžného vývojáře a je zajímavou technologií pro tvorbu API sloužících ke komunikaci mezi službami.

Samotný návrh zhotovené aplikace řeší jednu z nejdůležitějších problematik spojenou s hlasovými asistenty a API pro chytré domácnosti. Každý z vybraných hlasových asistentů vyžaduje jiný formát hodnot z ovládacích prvků zařízení a koncept zhotovený pro tuto aplikaci tento problém úspěšně řeší.

Část práce zabývající se implementací aplikace nabízí náhled do pozadí této aplikace a principů v ní použitých. Mezi nejzajímavější části aplikace patří již zmíněný převod hodnot a tvoření zpráv pro hlasové asistenty a MQTT listener starající se o přijímání zpráv z chytrých zařízení.

Po analýze principů a API jednotlivých asistentů byla úspěšně vytvořena aplikace. Následně bylo ověřeno, že tato aplikace je schopna fungování v reálném prostředí a s reálnými zařízeními. Testování aplikace poukázalo i na fakt, že rozhraní chytrých zařízení má stále své limity a proto je pravděpodobné, že výrobci hlasových asistentů tato rozhraní budou měnit a zlepšovat. Nyní je nutné nedostatky API pro chytrou domácnost kompenzovat pomocí konverzačního API.

Tabulka 3: Podporované ovládací prvky hlasových asistentů. Zdroj: [autor]

Ovládací prvky Amazon Alexa	**Ovládací prvky Google Assistant**
Alexa.BrightnessController	action.devices.traits.Brightness
Alexa.ColorController	action.devices.traits.ColorSetting
Alexa.PercentageController	action.devices.traits.HumiditySetting
Alexa.PowerController	action.devices.traits.OnOff
Alexa.TemperatureSensor	action.devices.traits.TemperatureControl
Alexa.ThermostatController	action.devices.traits.TemperatureSetting

Aplikace nyní podporuje pouze část z možných ovládacích prvků chytrých zařízení, jejíž výčet je možné vidět v tabulce 3. Mezi budoucí rozšíření aplikace patří definice všech ovládacích prvků podporovaných vybranými hlasovými asistenty. Dále může být aplikace rozšířena o uživatelské rozhraní tvořené pomocí JavaScriptových frameworků, ke kterému má uzpůsobené REST API.

Zhotovenou aplikaci je možné využít k přidání podpory vybraných hlasových asistentů pro DIY (Do It Yourself) chytrá zařízení. Mezi další využití aplikace spadá vývoj chytrých zařízení. Díky této aplikaci je možné otestovat koncept a funkčnost těchto zařízení. Aplikace je open source, díky tomu je možné na jejím základě, nebo pomocí principů v ní použitých vybudovat vlastní aplikaci, nebo službu pro výrobce chytrých zařízení.

Odkazy

1. *Define APIs for Alexa Conversations* [online]. 2021 [cit. 2021-02-17]. Dostupné z: https://developer.amazon.com/en-US/docs/alexa/conversations/define-apis.html.
2. *Conversational Actions* [online]. 2020 [cit. 2021-02-17]. Dostupné z: https://developers.google.com/assistant/conversational/overview.
3. *Understand the Smart Home Skill API* [online]. 2021 [cit. 2021-02-10]. Dostupné z: https://developer.amazon.com/en-US/docs/alexa/smarthome/understand-the-smart-home-skill-api.html.
4. *Smart Home Overview* [online]. 2021 [cit. 2021-02-17]. Dostupné z: https://developers.google.com/assistant/smarthome/overview.
5. *SPEAKER INDEPENDENT CONNECTED SPEECH RECOGNITION* [online]. 2007 [cit. 2021-02-17]. Dostupné z: http://www.fifthgen.com/speaker-independent-connected-s-r.htm.
6. ŠEVČÍK, Ondřej; OSOVSKÝ, Martin. *Zobrazení mezi fonologickými komponenty* [online]. 2006 [cit. 2021-03-01]. Dostupné z: https://digilib.phil.muni.cz/bitstream/handle/11222.digilib/100055/A_Linguistica_54-2006-1_3.pdf.
7. SHERIDAN, Noah. *A [Mathematical] Analysis of Sample Rates and Audio Quality* [online]. 2018 [cit. 2021-03-01]. Dostupné z: https://blogs.umass.edu/Techbytes/2018/04/20/a-mathematical-analysis-of-sample-rates-and-audio-quality/.
8. ORSÁG, Filip. *Some basic techniques of the speech recognition* [online]. 2002 [cit. 2021-03-01]. Dostupné z: https://www.researchgate.net/profile/Filip-Orsag/publication/312187923_Some_basic_techniques_of_the_speech_recognition/links/5874e69b08aebf17d3b3bdbb/Some-basic-techniques-of-the-speech-recognition.pdf.
9. *Language Processing Pipelines* [online]. 2021 [cit. 2021-02-10]. Dostupné z: https://spacy.io/usage/processing-pipelines.
10. MANNING, Christopher D.; RAGHAVAN, Prabhakar; SCHÜTZE, Hinrich. *Introduction to Information Retrieval*. Cambridge University Press, 2008. Dostupné také z: https://nlp.stanford.edu/IR-book/.
11. OCZKO, Jakub. *TEXT TO SPEECH SYSTEM* [online]. 2006 [cit. 2021-02-10]. Dostupné z: https://www.fekt.vut.cz/conf/EEICT/archiv/sborniky/EEICT_2006_sbornik/02-Magisterske_projekty/08-Grafika_a_multimedia/04-xoczko00.pdf.
12. GREENWALD, Will. *Amazon Echo Show vs. Google Nest Hub: Which Smart Display Should Be in Your Home?* [Online]. 2021 [cit. 2021-03-01]. Dostupné z: https://www.pcmag.com/news/amazon-echo-show-vs-google-nest-hub-smart-display-showdown.
13. *OAuth 2.0* [online]. 2018 [cit. 2021-02-10]. Dostupné z: https://docs.oracle.com/cd/E82085_01/160023/JOS%20Implementation%20Guide/Output/oauth.htm.

14. *Account Linking Concepts for Alexa Skills* [online]. 2021 [cit. 2021-02-10]. Dostupné z: https://developer.amazon.com/en-US/docs/alexa/account-linking/account-linking-concepts.html.

15. *Account linking* [online]. 2020 [cit. 2021-02-10]. Dostupné z: https://developers.google.com/assistant/identity.

16. *Conversational Actions Webhooks* [online]. 2021 [cit. 2021-02-10]. Dostupné z: https://developers.google.com/assistant/conversational/webhooks.

17. *Intent fulfillment* [online]. 2021 [cit. 2021-02-10]. Dostupné z: https://developers.google.com/assistant/smarthome/develop/process-intents.

18. *What is Python? Executive Summary* [online]. 2021 [cit. 2021-03-01]. Dostupné z: https://www.python.org/doc/essays/blurb/.

19. *Flask web development, one drop at a time* [online]. 2010 [cit. 2021-03-01]. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/.

20. *A pure-Python, gevent-friendly WSGI server* [online]. 2019 [cit. 2021-03-01]. Dostupné z: http://www.gevent.org/api/gevent.pywsgi.html.

21. *WTForms* [online]. 2008 [cit. 2021-03-01]. Dostupné z: https://wtforms.readthedocs.io/en/2.3.x/.

22. *Flask-Security* [online]. 2021 [cit. 2021-03-01]. Dostupné z: https://flask-security-too.readthedocs.io/en/stable/.

23. *Welcome to Flask-RESTX's documentation!* [Online]. 2020 [cit. 2021-03-01]. Dostupné z: https://flask-restx.readthedocs.io/en/latest/.

24. *What is Pony ORM?* [Online]. 2020 [cit. 2021-03-01]. Dostupné z: https://docs.ponyorm.org/.

25. *Public MQTT Broker* [online]. 2021 [cit. 2021-03-01]. Dostupné z: https://www.hivemq.com/public-mqtt-broker/.

26. *Getting Started on Heroku with Python* [online]. 2021 [cit. 2021-03-01]. Dostupné z: https://devcenter.heroku.com/articles/getting-started-with-python.

27. *Use the Smart Home Tests* [online]. 2021 [cit. 2021-03-01]. Dostupné z: https://developer.amazon.com/en-US/docs/alexa/smarthome/test-automated.html.

28. *Test and share Smart Home Actions* [online]. 2021 [cit. 2021-03-01]. Dostupné z: https://developers.google.com/assistant/smarthome/develop/testing.

29. *LOLIN D1 mini* [online]. 2019 [cit. 2021-03-01]. Dostupné z: https://www.wemos.cc/en/latest/d1/d1_mini.html.

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2019/2020

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení: **David Tláskal**
Osobní číslo: **I1700713**
Adresa: Býšť 200, Býšť, 53322 Býšť, Česká republika

Téma práce: Využití hlasových asistentů v chytrých domácnostech
Téma práce anglicky: Evaluation of voice assistants in smart homes

Vedoucí práce: Mgr. Hana Rohrová
Katedra informačních technologií

Zásady pro vypracování:

Cílem práce je představit vybrané domácí hlasové asistenty a zhodnotit jejich možnosti z hlediska využití v chytrých domácnostech. Zhodnocení bude provedeno vytvořením testovacích aplikací ve zvoleném programovacím jazyce.
Osnova:
1. Úvod
2. Cíle práce
3. Principy rozpoznávaní hlasu
4. Hlasoví asistenti
5. Návrh aplikací
6. Implementace aplikací
7. Testování
8. Závěr

Seznam doporučené literatury:

Fundamentals of Speaker Recognition 978-1489979223
REST API Design Rulebook 978-1449310509
Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming 978-1593279288

Podpis studenta: Datum:

Podpis vedoucího práce: Datum: