Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Česká hlasová asistentka Klára

Bakalářská práce

Autor: Dominik Palla

Studijní obor: Aplikovaná informatika

Vedoucí práce: prof. RNDr. PhDr. Antonín Slabý, CSc.

Hradec Králové Duben 2020

Prohlášení

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

V Trutnově dne 20. 4. 2020

Poděkování

Děkuji prof. RNDr. PhDr. Antonínu Slabému, CSc. za vedení této bakalářské práce a jeho velice cenné rady.

Anotace

Tato práce se zabývá tématikou mobilních hlasových asistentů s umělou inteligencí (nebo její simulací). V první části seznámíme čtenáře s metodami přístupu k tomuto typu softwaru, vysvětlíme si jeho obecný princip a představíme si nejznámější a nejúspěšnější projekty této kategorie. Probereme tedy technologii rozpoznávání řeči a jejího převodu na text, možnosti řešení samotné „inteligentní“ složky aplikace, a nakonec hlasovou syntézu. Ve druhé části se zaměříme na implementaci vlastního asistenta (respektive asistentky – Kláry) se simulací umělé inteligence pomocí Levenshteinova algoritmu, a to pro platformu iOS.

Klíčová slova: bakalářská práce, iOS, hlasový asistent, umělá inteligence, rozpoznání řeči, hlasová syntéza, Levenshteinův algoritmus, programovací jazyk Swift, programovací jazyk PHP, software pro nevidomé

Annotation

Title: Czech voice assistant Clara

This bachelor thesis deals with the topic of mobile voice assistants with artificial intelligence (or its simulation). In the first part, we will show the reader the methods of implementation of this type of software, explain its general principle and present the best known and most successful projects in this category. So, we will discuss the technology of speech recognition and its conversion to text, the possibilities of solving the "intelligent" component of the application itself, and finally voice synthesis. In the second part, we will focus on the implementation of our own assistant (called Clara) with the simulation of artificial intelligence using the Levenshtein algorithm. Our implementation will be done for the iOS platform.

Keywords: bachelor thesis, iOS, voice assistant, artificial intelligence, speech recognition, voice synthesis, Levenshtein's algorithm, Swift programming language, PHP programming language, software for the blind

Obsah

SEZNAM OBRÁZKŮ

1 ÚVOD A CÍL PRÁCE

Přirozenou vlastností každého z nás je snaha o zjednodušení každodenních činností. Tento fakt a ohromný pokrok ve vývoji moderních technologií vede řadu technologických gigantů k tvorbě hlasových asistentů. Tito giganti pracují na poli umělé inteligence a jejich snahou je přinést světu zcela autonomního agenta, který by měl být schopen víceméně porozumět přirozenému jazyku člověka a v této formě přirozeného jazyka také řádně odpovědět. Měl by fungovat jako náš nerozlučný pomocník téměř ve všech aspektech našeho života. Hlasoví asistenti za nás vytvářejí události, nákupní seznamy, píší emaily a textové zprávy, zjišťují informace, připomínají nám úkoly. Největší přínos těchto technologií je zcela jistě v podpoře handicapovaných uživatelů, zejména pak nevidomých. Přesto převládá zájem spíše o usnadnění práce nehandicapovaným uživatelům, např. formou chytrých domácností, které v této době zažívají obrovský boom.

Cílem této práce bude vývoj takového jednoduššího agenta pro platformu iOS, který se bude snažit tuto umělou inteligenci alespoň simulovat. Toto téma jsem zvolil z toho důvodu, že jsem se před několika lety tímto problémem zabýval. Jednoho takového chytrého agenta jsem vytvořil pro platformu Android, a to za spolupráce komunity nevidomých. Aplikace se stala mezi takto handicapovanými lidmi v ČR celkem populární, používalo je přes 10.000 uživatelů.

2 SOUČASNÁ ŘEŠENÍ HLASOVÝCH ASISTENTŮ

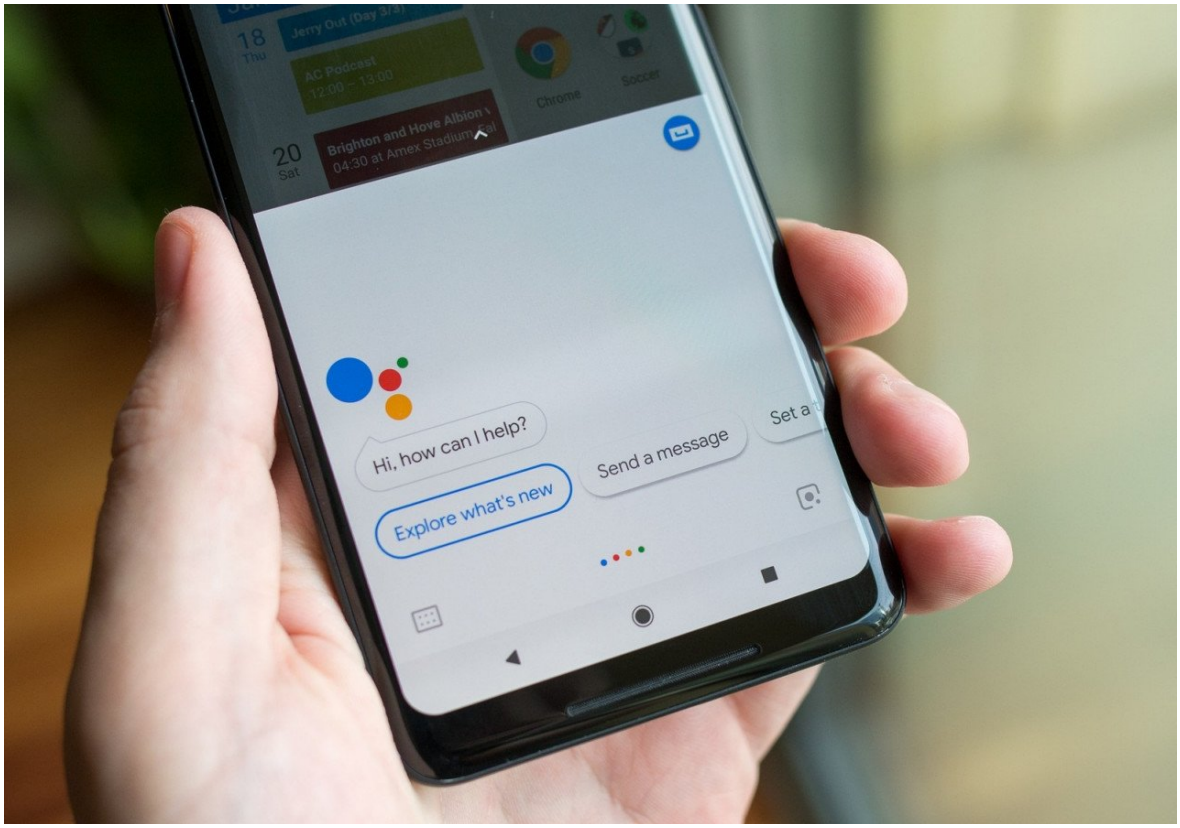
Hlasových asistentů existuje na trhu mnoho. V této kapitole budou probrány ty nejznámější z nich. Všechna uvedená řešení asistentů fungují na stejném základním principu. V prvním kroku je třeba implementovat rozpoznávání mluvené řeči a její následné převedení do textové podoby. K tomu využíváme technologii voice recognition. Následně je třeba implementovat „mozek“ služby, tedy nějaký algoritmus, který dokáže z textové podoby vytáhnout logickou informaci, kterou text reprezentuje, nejčastěji otázku. Tyto algoritmy se společnost od společnosti liší, mnohdy jsou utajovány. Obecně existuje mnoho různých postupů, jak tento problém řešit. Nakonec už jen zbývá provést hlasový výstup, čehož je dosaženo pomocí technologie TextToSpeech (hlasový syntetizér), která výsledný textový řetězec (odpověď softwaru) převede na mluvenou řeč. Více se těmto jednotlivým technologiím a algoritmům budu věnovat v následujících kapitolách.

2.1 Google Assistant

Asi nejznámější a nejrozšířenější řešení od společnosti Google, které běží pod operačním systémem Android. Asistenta je možné využívat na telefonu, tabletu, počítači, chytrých hodinkách, chytrém autě, chytrém reproduktoru (Google Home) nebo chytrém displeji (Smart Display).[1] Asistent je dostupný v 16 světových jazycích, bohužel čeština mezi nimi není, přestože na konferenci I/O v květnu roku 2018 bylo oznámeno, že do konce roku 2018 bude podporována. Při vývoji tohoto asistenta Google vsadil na programovací jazyk C++. První vydání aplikace proběhlo v květnu roku 2016. Do té doby existovala podobná služba Google Now, která však neumožňovala obousměrnou komunikaci (tedy mezi uživatelem a softwarem).

V prosinci roku 2016 Google umožnil vývojářům vytvářet vlastní aplikace pro hlasového asistenta (Actions on Google). V roce 2017 dokonce přidal podporu a nástroje pro vytváření různých her v rámci asistenta.

V květnu 2018 Google představil technologii Google Duplex, rozšíření Asistenta Google, které mu umožňuje provádět přirozené konverzace napodobováním lidského hlasu, a to způsobem, který je podobný robocallingu.[2] Asistent může samostatně plnit úkoly, jako je volání do kadeřnictví za účelem objednání schůzky, plánování a rezervace restaurací nebo zavolání firmám, aby ověřily otevírací hodiny obchodního domu.[3] Kromě toho, že Duplex dokáže většinu svých úkolů dokončit zcela autonomně, umí rozpoznat také situace, kdy nedokáže úkol dokončit, a může tak signalizovat lidskému operátorovi, aby mu s tím pomohl. Duplex byl vytvořen tak, aby mluvil přirozenějším hlasem a jazykem začleněním řečových disfluencí, jako jsou výplňová slova jako „hmm“ a „uh“, a používáním běžných frází jako „mhm“ a „gotcha“ spolu s více lidskou intonací a latencí odpovědí. [4] [5] [6] Duplex je v současné době ve vývoji, avšak majitelé telefonů Google Pixel žijící v Atlantě, New Yorku, Phoenixu či San Franciscu mají od konce roku 2018 jeho omezené vydání. Jsou tak schopni tuto technologii využívat, ale prozatím jen k rezervaci restaurací. [7]



Obrázek 1 Google Assistant

2.2 Siri

U uživatelů iOS, macOS, iPadOS, watchOS či tvOS od společnosti Apple je jasnou (a jedinou) volbou asistentka Siri. Za ní nestojí nikdo jiný než výše zmíněná společnost, přestože původně byla vyvinuta Mezinárodním umělým zpravodajským střediskem SRI (odtud název Siri). Projekt byl financován z DARPA a zakladateli byli Dag Kittlaus, Harry Saddler, Adam Cheyer a Tom Gruber (ze SRI).

Hlasový asistent byl vypuštěn jako aplikace pro systém iOS v únoru 2010. Společnost Apple aplikaci o dva měsíce později získala.

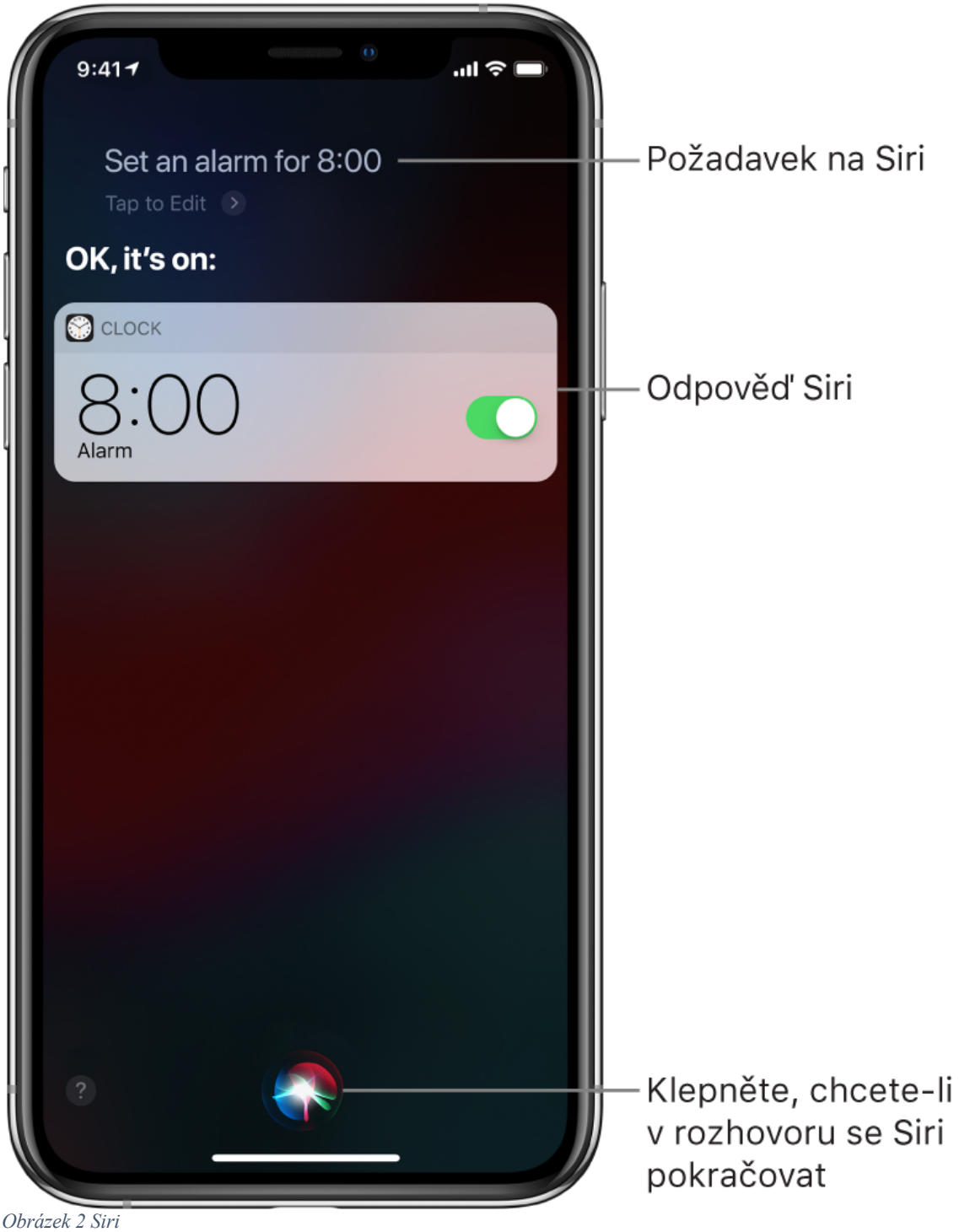
Asistentka Siri byla následně v říjnu 2011 začleněna do iPhone 4S a stala se tak navždy nedílnou součástí produktů společnosti Apple, která se v průběhu let přizpůsobila i ostatním hardwarovým zařízením od iPhonů, iPodů, Maců a chytrých hodinek až po Apple TV.

Rozpoznávání řeči zajišťuje společnost Nuance Communications. Dlouhá léta to nebylo oficiálně uznáváno společnostmi Apple ani Nuance, dokud CEO společnosti Nuance – Paul Ricci - nepotvrdil informace na konferenci o technologii v roce 2011.

Od roku 2017 Siri mluví 21 jazyky s lokalizací pro 36 zemí.[8] Čeština mezi nimi bohužel není. Proto je Siri i v ČR využívána v angličtině. Z uniklých zdrojových kódů iOS 8 však bylo patrné, že další jazyky v čele s češtinou jsou ze strany Applu plánovány, protože mají již zakomponovanou nativní podporu v systému. Siri se postupně učí další jazyky pomocí funkce diktování textu, která funguje na zařízeních od Applu v češtině již řadu let.

Asistentka využívá pokročilé technologie pro učení a podporuje širokou škálu uživatelských příkazů, včetně provádění telefonních akcí, zjišťování základních informací, plánování událostí a připomenutí, ovládání nastavení zařízení, vyhledávání na internetu, navigaci, hledání informací o zábavě a má schopnost pracovat s aplikacemi integrovanými v iOS.[9]

Při vydání iOS 10 (v roce 2016) společnost Apple otevřela omezený přístup třetích stran ke službě Siri pro zasílání zpráv, plateb, sdílení jízd a aplikací pro internetové volání.



Obrázek 2 Siri

2.3 Cortana

Cortana je virtuální asistentka, která je součástí operačních systémů Windows Phone 8.1 a Windows 10 od společnosti Microsoft. Nyní je nově dostupná také na herních konzolích Xbox One.

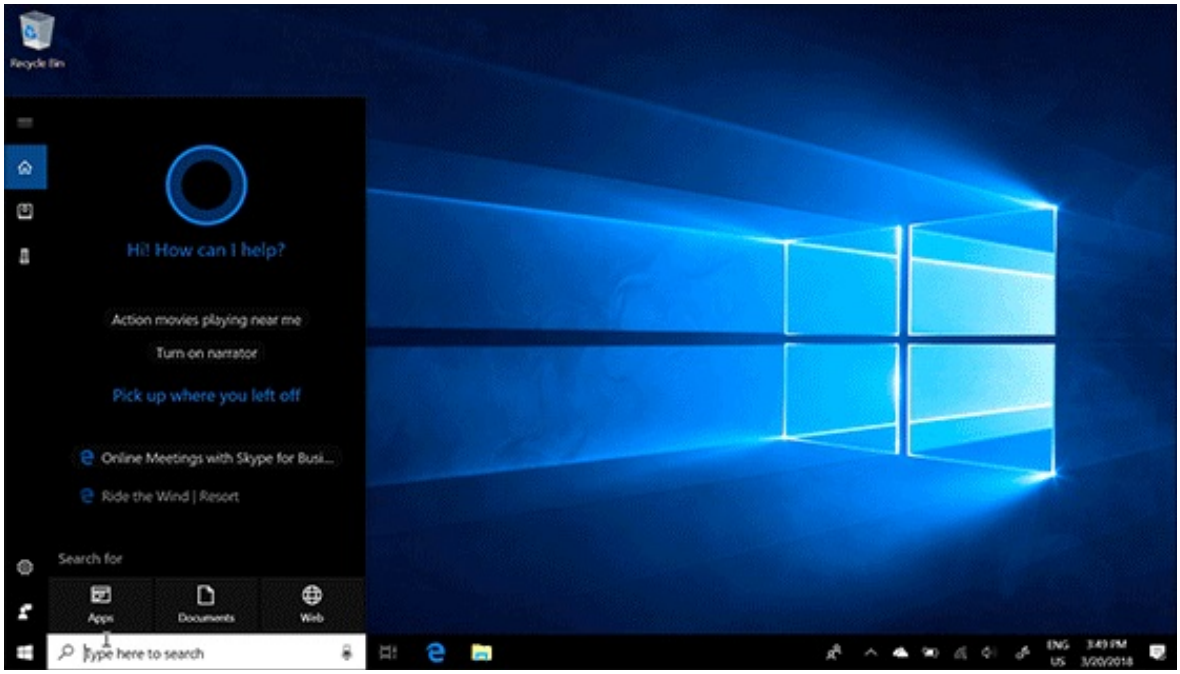
Cortana byla poprvé představena Joem Belfiorem na konferenci Microsoft BUILD 2014 v San Franciscu (2014). Název Cortana je odvozen od fiktivní postavy ze série her Halo. Její hlas patří, stejně jako v sérii Halo, Jen Taylorové.

Na základě hlasových povelů umí pracovat s kalendářem, spravovat seznamy kontaktů, otevírat aplikace, posílat e-maily a SMS, synchronizovat data mezi mobilním telefonem a počítačem apod.[10]

Cortana čerpá data z portálů Bing a Wikipedie nebo z aplikací od Microsoftu (např. Počasí nebo Finance). Dalším zdrojem informací jsou pro ni uživatelem používané aplikace v počítači nebo mobilu.

Asistentka od Microsoftu zatím rozumí jen v angličtině, čínštině (v čínských oblastech se jmenuje Xiao Na[11]) a od roku 2014 také italštině, francouzštině, němčině a španělštině[12]. Tyto verze byly nejprve dostupné jen ve verzi alpha, tedy je testovali sami uživatelé a Microsoft sbíral informace o používání. V těchto jazykových verzích Cortana zdaleka neumí všechno to, co umí její anglická verze.

Po představení Windows 10 (a tedy snaze sjednotit všechny platformy v jednu s jedním prostředím vhodným pro všechna zařízení) chce Microsoft také plnohodnotně využít asistentku Cortanu. Má být integrována právě do desktopu ve Windows 10 a bude s ní počítáno i v podnikové sféře.[13]



Obrázek 3 Cortana

2.4 Amazon Alexa

Amazon Alexa, známá jednoduše jako Alexa, je virtuální pomocná umělá inteligence vyvinutá společností Amazon, poprvé použitá v inteligentních reproduktorech Amazon Echo vyvinutých společností Amazon Lab126.

Je schopna hlasové interakce, přehrávání hudby, vytváření seznamů úkolů, nastavení alarmů, streamování podcastů, přehrávání zvukových knih a poskytování informací o počasí, dopravě, sportu a dalších informacích v reálném čase, jako jsou např. zprávy.[14]

Alexa může také ovládat několik inteligentních zařízení pomocí domácího automatizačního systému. Uživatelé mohou rozšířit možnosti Alexy instalací „dovedností“ (další funkce vyvinuté vývojáři třetích stran).

Většina zařízení s Alexou umožňuje uživatelům aktivovat zařízení pomocí „budícího“ slova (např. „Alexa“ nebo „Amazon“), ostatní zařízení (například mobilní aplikace Amazon pro iOS nebo Android a Amazon Dash Wand) vyžadují stisknutí tlačítka k zahájení poslechu.

V současné době jsou interakce a komunikace s Alexou k dispozici pouze v angličtině, němčině, francouzštině, italštině, španělštině[15], portugalštině, japonštině a hindštině.[16] V Kanadě je Alexa k dispozici v angličtině a francouzštině (s Quebeckým přízvukem).[17] [18]

V listopadu 2018 měl Amazon více než 10 000 zaměstnanců pracujících na Alexe a souvisejících produktech.[19] V lednu 2019 Amazon oznámil, že prodal více než 100 milionů zařízení podporujících asistentku Alexu.[20]

V září 2019 Amazon uvedl na trh mnoho nových zařízení a dosáhl mnoha rekordů při konkurenci světovému inteligentnímu domácímu průmyslu. Nové Echo Studio se stalo prvním chytrým reproduktorem se zvukem 360 a zvukem Dolby. Mezi další nová zařízení patřil Echo dot s hodinami, Amazon Echo třetí generace, Echo Show 8, Echo Flex, Echo Loop a další zařízení se zabudovanou Alexou (sluchátka, brýle, prsten).

Obrázek 4 Amazon Echo 2019 se zabudovanou Alexou

2.5 Antelli – české řešení

Antelli je český projekt vývojáře Štěpána Šonského. V době fungování mého řešení pro nevidomé byla Antelli mým jediným konkurentem.

Aplikace je postavena na Google Cloud API a nástrojích této služby programovat umělou inteligenci.

V podstatě nabízí základní funkce ostatních hlasových asistentek. Od roku 2018 dokonce umožňuje ostatním vývojářům využívat své API a doplňovat tak do Antelli nové funkce.

Nejnovější funkcí je schopnost pracovat s chytrou domácností od Xiaomi.[21]

Obrázek 5 Česká Antelli z dílny vývojáře Štěpána Šonského

2.6 ClaraSys (Android verze)

Jak už jsem se zmínil v úvodní kapitole, před několika lety jsem vytvořil pomocnou aplikaci (nejen) pro nevidomé, která pokrývala v podstatě veškeré funkce, jako konkurenční asistentky. Dával jsem v ní důraz na češtinu a na postavu české asistentky – Kláry.

Mezi nejoblíbenější funkci například patřilo chytré hlasové buzení. V podstatě to znamenalo, že Klára uživatele vzbudila svým hlasem a jakmile se ujistila, že je vzhůru, poskytla mu přehled událostí, které má na daný den naplánovány, předpověď počasí a výběr zpráv.

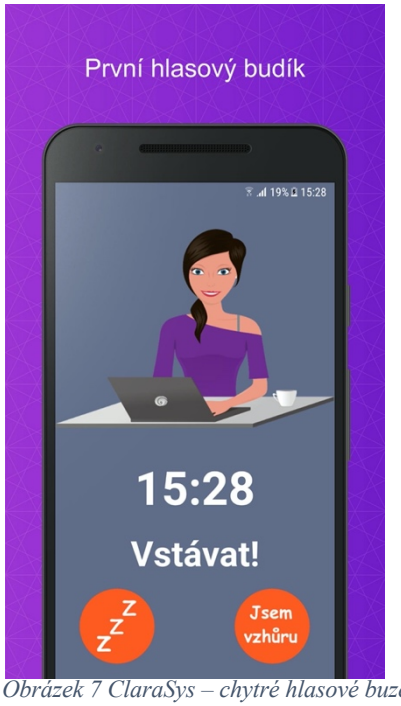
Původně jsem aplikaci vytvářel jen pro sebe a pro svoji chytrou domácnost, kterou jsem si vytvářel přes Arduino. Když jsem naučil Kláru ovládat na povel světla, zásuvky a garáž, u mých známých to vzbudilo takový ohlas, že jsem se rozhodl aplikaci publikovat veřejně.

Teprve po roce dostupnosti aplikace mezi přibližně tisíci uživateli mě kontaktovalo sdružení nevidomých, v čele s předsedou a zpěvákem Radkem Žaludem. Tak započala naše dlouholetá spolupráce, při které jsem se snažil přizpůsobit aplikaci právě nevidomým uživatelům, neboť jak jsme se s panem Žaludem shodli, v používání této technologie nevidomými je její skutečný potenciál.
Aplikace umožňovala uživatelům v případě nesprávné odpovědi navrhnout odpověď lepší, a tak se Klářina databáze znalostí a schopnosti stále rozšiřovaly.

Bohužel mě tento projekt stál příliš času a energie. Nakonec mě permanentní stres při práci, škole, souběžném vylepšování svého projektu a podpoře 10 tisíců uživatelů donutil projekt v roce 2018 ukončit.

Databáze znalostí (čítající přes 2000 frází), která mi zůstala, bude použita pro základ svého projektu v rámci této práce. Tentokrát však budu tento mechanismus implementovat na platformě iOS.

Obrázek 6 ClaraSys - výchozí obrazovka



Obrázek 7 ClaraSys – chytré hlasové buzení

3 Technologie rozpoznávání hlasu (voice recognition)

Technologie umožňující rozpoznávat hlasový vstup a převádět ho na text hraje v éře hlasových asistentů naprosto klíčovou roli. I sebelepší algoritmus s umělou inteligencí by selhal bez schopnosti spolehlivého rozpoznání hlasu.

3.1 Základní princip

Před deseti lety bylo možné aplikovat tuto technologii ve dvou variantách: diskrétní a spojité rozpoznávání. Diskrétní varianta vyžadovala výrazné mezery mezi jednotlivými slovy, takže bylo pro software jednodušší oddělit od sebe jednotlivá slova a ty pak převést do psaného textu. Protože je však taková mluva velice nepřirozená a zdlouhavá, bylo vyvinuto spojité rozpoznávání hlasu. Díky tomu může uživatel mluvit větami s běžnými mezerami mezi slovy, čímž je diktování mnohem rychlejší a příjemnější. V podstatě všechny moderní systémy jsou tedy dnes založeny na spojitém rozpoznávání, což v důsledku u hlasových asistentů přispívá ke zdání normálního rozhovoru mezi dvěma lidmi, přestože mluvíme s aplikací.

Současné programy lze rozdělit ještě podle dalšího klíče na dvě kategorie:

3.1.1 Malý slovník – hodně uživatelů

Tato varianta se nejčastěji uplatňuje v automatizovaných telefonních automatech (IVR). Program má sice velice malou slovní zásobu, ale za to slova bezpečně rozpozná u širokého spektra uživatelů, tedy hlasů. Je toho dosaženo vysokou tolerancí odchylek mezi jednotlivými slovy.

3.1.2 Velký slovník – málo uživatelů

Oproti tomu tato druhá varianta je využívána například v podnikové sféře, kde se systémem pracuje poměrně úzké spektrum uživatelů (a tedy hlasů). Přesnost se pohybuje okolo 85 %, ale lze ji trénovat pro konkrétního uživatele. S novým uživatelem, který má výrazně odlišný přízvuk či řečový vzor, bude úspěšnost prudce klesat. Díky tomuto omezení je však možné systém využít pro rozpoznání obrovského množství různých slov.

3.2 Postup převodu řeči na text

Během celého komplikovaného postupu je třeba, aby program prošel několika komplexními kroky.

Při mluvení člověk vytváří vibrace ve vzduchu. Prvním krokem je tedy digitalizace vstupního signálu. Na vstupu ADC probíhá transformace vstupních analogových vln na diskrétní data. Ta jsou získávána odebíráním hodnot těchto analogových vln v pravidelných intervalech.

V dalším kroku program filtruje digitalizovaný zvuk tak, aby odstranil nežádoucí hluk v pozadí nebo aby se zvuk rozdělil na jednotlivá frekvenční pásma, která se budou následně zpracovávat zcela samostatně. Některá frekvenční pásma se musí vypustit zcela, protože v

nich člověk neslyší a ani v nich nemluví. Dále v tomto kroku také probíhá normalizace digitalizovaných dat a upravuje se hlasitost do rozsahu daného vzorovými daty. Protože lidé mluví různou rychlostí, jsou data upravena tak, aby jejich rychlost odpovídala těm datům, která jsou v tomto systému uložena jako vzorová a se kterými budou následně porovnávána.

Tento digitalizovaný signál je poté rozdělen na malé dílky, které jsou krátké pouze několik setin sekundy. Program následně porovná tyto vzorky se známými fóny v daném jazyce. Fóny jsou zvuky, které představují určité hlásky. Jejich složením získáme smysluplné výrazy. Zatímco anglická abeceda má 26 písmen (ovšem 40 různých foném), v češtině máme písmen 42 (ale jen cca 39 různých foném).

Následující krok se nám může zdát snadný. Vlastně stačí porovnat vyřčený foném s fonémy, které se používají v konkrétním jazyce. Poté stačí najít ten nejpodobnější a na základě toho sestavit celé slovo. Ve skutečnosti je postup ovšem o poznání složitější, ale princip se neliší.

Software zkoumá každý foném v kontextu okolních fonémů a hledá nejpravděpodobnější kombinace na základě komplexního statistického modelu. Ten je založený na velké knihovně známých slov v konkrétním jazyce. Program je poté na tomto základě schopen určit slovo, které uživatel řekl nejpravděpodobněji.

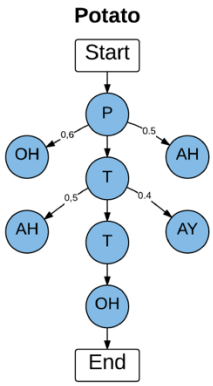
Dnes však navíc je dosaženo ještě větší přesnosti, protože program dává do kontextu každé nalezené slovo se slovy, která mu předcházela. Na základě toho je schopen přehodnotit veškerá nalezená slova a v případě, že fonémům odpovídají také jiná možná slova, která jsou však dle kontextu pravděpodobnější, zvolí je.[22]

3.3 Metody statistického modelování

V dnešní době tyto systémy využívají rozsáhlé a výkonné statistické modely, které jim pomohou určit nejpravděpodobnější výsledek. Nejčastějšími modely v tomto směru je Hidden Markov Model (HMM) nebo neuronové sítě. Tyto systémy zahrnují složité matematické a statistické funkce, pomocí kterých se z informací, které systém zná, zjišťují informace, které zatím nezná.[23]

Nejčastějším používaným je právě HMM model. V tomto modelu jsou pozorovateli známy nejen výstupy modelu, ale také jeho vnitřní hlasy. U tohoto modelu jsou stavy nepozorovatelné, ovšem výstup modelu pozorovatelný je. Na výstup má přitom pravděpodobností vliv každý vnitřní stav.

To, že je tento model skrytý (hidden), vlastně znamená, že není známa posloupnost vnitřních stavů, které vedou k výstupu, přestože parametry modelu známé jsou. Na základě tohoto modelu dokáže program předpovědět nejpravděpodobnější následující foném. Hranám doprovodného grafu během tohoto procesu program přiřazuje ohodnocení na základě slovníku. Pro rozpoznávání sousloví a vět je proces ještě složitější, a to zejména proto, že program musí najít začátek a konec slova. Příkladem z angličtiny jsou sousloví „recognize speech“ a „wreck a nice beach“.[24]



Obrázek 8 HMM model slova „potato" s ohodnocenými hranami (Převzato z [24])

Složitost tohoto problému je v tom, že slovník programu obsahuje typicky tisíce slov daného jazyka (nejčastěji až 60 000 slov). Když budeme hledat 3 po sobě jdoucí slova, tak máme 216 biliónů možností. Prohledat bez další znalosti všechny možnosti, které nám slovník poskytuje, bude i tomu nejvýkonnějšímu počítači trvat dlouho.

K tomu nám slouží tréninkové programy, ve kterých program dostane tisíce hodin mluveného slova společně se správným výsledkem. Na základě těchto dat se následně vytvoří akustické modely slov a pravděpodobnostní sítě pro sousloví a věty. Někdy se software musí od koncového uživatele doučit. Uživatel může být vyzván k přečtení několika vět. Systém se díky tomu přizpůsobí řeči uživatele a její hlasitosti, rychlosti a intonaci.

HMM model nachází využití mimo jiné také v rozpoznávání gest a ručně psaného textu.

3.4 Dostupné API pro rozpoznávání řeči

Na trhu existuje mnoho řešení od různých společností. Mezi ta nejznámější patří Google Cloud Speech API, Bing speech API, IBM Watson Developer Cloud Speech to Text, Kaldi anebo Apple Speech recognizer, který budu implementovat v následujících kapitolách.

3.5 Technologie použitá v této práci

V rámci této práce budeme implementovat technologii rozpoznávání řeči Apple Speech recognizer. Je to logické rozhodnutí, když přihlédneme k tomu, že naši testovací platformou je Apple iOS.

4 Způsoby řešení problému umělé inteligence

Dostáváme se k nejdůležitější části celé problematiky. Nyní se zaměříme na způsoby zpracování textu, který jsme získali metodou popsanou v předchozí kapitole.

Existuje několik možností, kterými můžeme textová data zpracovat. Obecně lze tyto postupy rozdělit do 2 kategorií.

První kategorií je zachycení klíčových slov, větných vzorců a sekvencí.[25] Tato metoda je jednodušší a bude použita v rámci této práce.

Druhou kategorií je neuronová síť, která potřebuje ke správné funkci velké množství dat ke zkoumání, aby se mohla učit a zdokonalovat.[26] Tato data můžeme získat právě postupem z první kategorie. Zde již můžeme mluvit o problematice opravdové umělé inteligence.

4.1 Definice umělé inteligence

Umělá inteligence (Artificial Intelligence, AI) je vědní obor, který se zabývá tvorbou strojů, které vykazují inteligenci. Přesná a jediná definice bohužel neexistuje, neustále se vedou spory, jak umělou inteligenci nejlépe definovat. Definic existuje mnoho, ale vždy se jedná pouze o definice obecné, které nevysvětlují, co vůbec umělá inteligence je.[27]

Zatím nejlepší definici vytvořil Marvin Minsky v roce 1967:

„Umělá inteligence je věda o vytváření strojů nebo systémů, které budou při řešení určitého úkolu užívat takového postupu, který, kdyby ho dělal člověk, bychom považovali za projev jeho inteligence.“[28]

4.2 Umělá inteligence podle Turinga

Tento test vytvořil Alan Turing v roce 1950. Má za úkol vyhodnotit, zda se daný stroj chová opravdu inteligentně či nikoli. Protože inteligenci nelze přesně definovat, porovnává se v tomto testu stroj s člověkem.

Test probíhá tak, že máme 2 místnosti. V jedné místnosti sedí testující člověk. Ve druhé je umístěn jiný člověk a testovaný předmět, např. počítač s „inteligentním“ programem. Testující člověk pokládá otázky v přirozené řeči a předává je do druhé místnosti. Tam na ně náhodně odpoví buď člověk nebo počítač formou tištěné odpovědi. V případě, že testující není schopen rozpoznat, která odpověď je od člověka a která od počítače, pak můžeme hovořit o tom, že stroj je skutečně inteligentní.

Turingův test byl nejpoužívanějším ukazatelem inteligence strojů, nakonec se však prokázala přílišná subjektivnost hodnocení.[29]

4.3 Metoda zachycení klíčových slov a větných vzorců

Ke zjištění podobnosti dvou textových řetězců potřebujeme algoritmus, který nám tuto podobnost číselně ohodnotí. Vlastně potřebujeme vhodnou metriku k výpočtu tzv. editační vzdálenosti mezi nimi.

Editační vzdálenost kvantifikuje podobnost dvou textových řetězců. Označuje nám počet změn, které potřebujeme k transformaci jednoho řetězce na druhý.

Jako změna se přitom uvažuje vkládání, odstranění nebo záměna znaku.

Pojem editační vzdálenost (Edit distance[30][31][32][33]) se často používá přímo pro Levenshteinovu vzdálenost (viz kapitola 4.3.1), která je neznámějším a nejčastějším řešením.

Ve skutečnosti však tento pojem označuje skupinu metrik, sloužící k určení editační vzdálenosti.

Příklady algoritmů [34]:

- Levenshteinova vzdálenost (viz kapitola 4.3.1)
- Hammingova vzdálenost
- Damerau–Levenshteinova vzdálenost
- Jaro vzdálenost
- Jaro-Winkler vzdálenost

4.3.1 Levenshteinova vzdálenost

Mějme dva řetězce (sekvence znaků) A a B. Vzdálenost mezi nimi je určena jako minimální množství nutných změn, které musíme učinit v sekvenci A, aby se z ní stala přesná kopie sekvence B. Přitom se jako změna počítá vkládání, odstranění nebo záměna znaku.[35][36]

Vztah pro výpočet Levenshteinovy vzdálenosti pro dva řetězce A a B (s délkami ||a|| a ||b||) je dán vztahem $lev_{A,B}(||a||, ||b||)$.

S využitím rekurze můžeme funkci pro výpočet této vzdálenosti definovat následovně:

$$lev_{A,B}(i,j) = \begin{cases} \max(i,j), & když \min(i,j) = 0; \\ \min \begin{cases} lev_{A,B}(i-1,j)+1 \\ lev_{A,B}(i,j-1)+1 \\ lev_{A,B}(i-1,j-1)+1_{A_i \neq B_j} \end{cases}, & jindy \end{cases}$$

Obrázek 9 Vzorec Levenshtainovy vzdálenosti

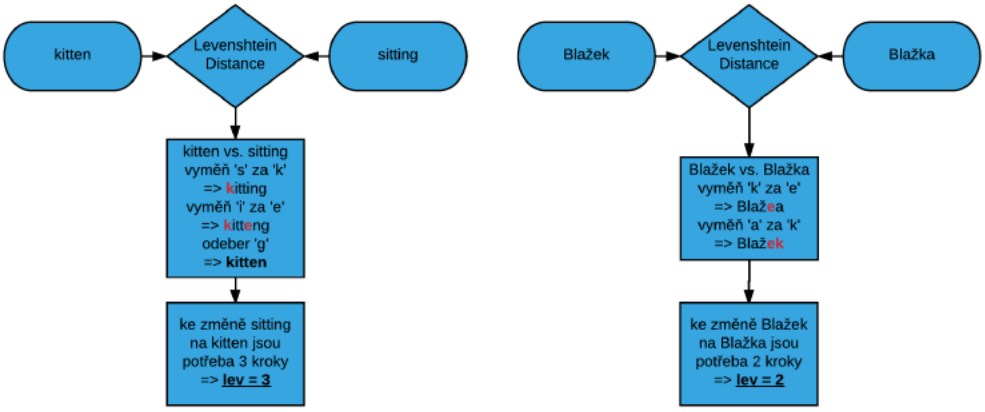
První element v minimu označuje možnost odstranit znak.
Druhý element vyplývá z možnosti přidání znaku.
Poslední, třetí element vyjadřuje (ne)shodu znaků na daných pozicích.

$lev_{A,B}(i, j)$ je vzdálenost mezi i-tým znakem řetězce A a j-tým znakem řetězce B. Platí, že:

$$1_{A_i \neq, B_j} = \begin{cases} 0, & když \quad A_i = B_j \\ 1, & jindy \end{cases}$$

Obrázek 10 Podmínky vzorce

Na obrázku č. 11 je ukázka Levenshteinova algoritmu formou diagramu.[37]



Obrázek 11 Ukázkové schéma [37]

4.4 Neuronové sítě

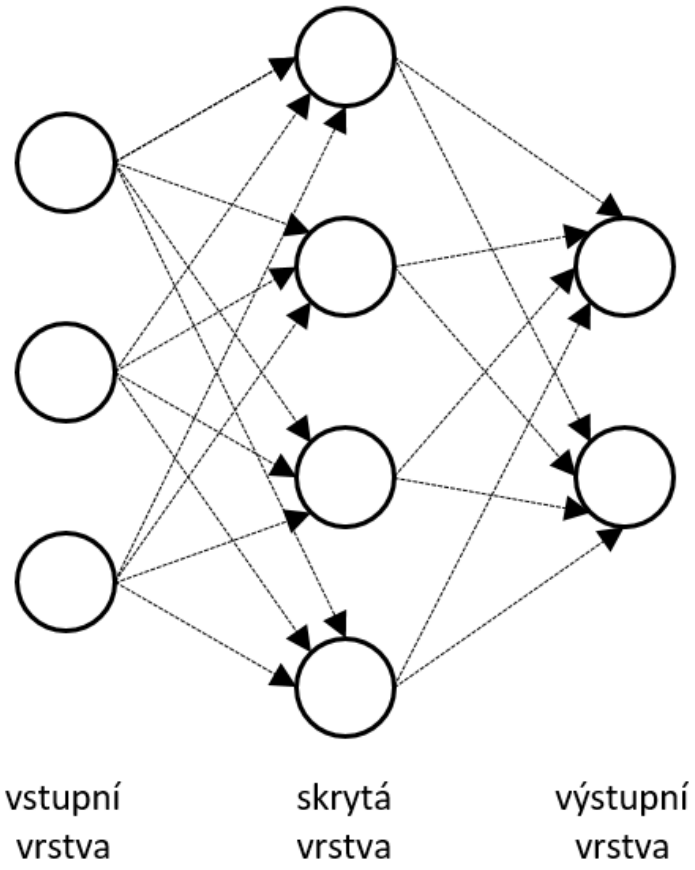
Neuronové sítě v umělé inteligenci jsou inspirovány biologickými strukturami. Oproti nim jsou však značně zjednodušeny. Jednak z důvodu zrychlení výpočtů, a také používají jiné mechanismy učení, než jaké používá mozek. Cílem informatiky není věrně simulovat biologické struktury, ale především řešit praktické problémy. Touto tématikou se zabývají právě výpočetní neurovědy.

Základním prvkem přirozené i umělé neuronové sítě je neuron (perceptron). Neurony jsou navzájem propojeny a předávají si signály. Každý neuron může mít více vstupů, ale jen jeden výstup (tento výstup ale může být poslán více než jednomu dalšímu neuronu). Vstupem neuronu může být buď výstup z jiného neuronu nebo informace z vnějšku. Každý vstup má určitou váhu.

Jakmile neuron přijme vstupy, tak jejich hodnoty vynásobí odpovídajícími váhami. Následně tato čísla sečte, pokud je výsledek větší než stanovený práh, výsledek se transformuje předem danou přenosovou funkcí a pošle na výstup.

Pro řešení jednoduchých úloh by stačil i jediný neuron. Většinou jich ale použijeme více a vzájemně je propojíme.

Na obrázku č. 12 vidíme síť typu 3-4-2. Máme zde tři vstupy. Vstupní vrstva jen zopakuje hodnoty vstupů a předá je všem neuronům v tzv. skryté vrstvě. Jejich výstupy jsou vstupem pro dva další neurony ve výstupní vrstvě. Hodnoty výstupů z těchto neuronů pak kódují výstup celé neuronové sítě.



Obrázek 12 Síť typu 3-4-2

4.4.1 Učení neuronové sítě

Máme-li pevně danou strukturu sítě (tedy počty neuronů ve vrstvách a jejich spojení) a transformační funkci, tak to, jak síť vstupy převede na výstupy, závisí na hodnotě vah a prahů. Tyto hodnoty se přitom stanovují v procesu, kterému se říká učení. Existují dva druhy učení umělých neuronových sítí – učení s učitelem a učení bez učitele.

4.4.2 Učení s učitelem

Máme množinu vstupů a požadovaných výstupů (např. sadu parametrů strojů se známou, člověkem diagnostikovanou závadou). Vstupy se propustí do sítě, výstup se následně porovná s požadovaným výstupem, a nakonec se provede korekce (změna vah a prahů), a to tak, aby byl rozdíl mezi skutečným a požadovaným výstupem co nejmenší. Síť se vlastně v tomto případě učí ze svých vlastních chyb.

Máme-li síť správně nastavenou a poskytneme jí k učení reprezentativní množinu vstupů a výstupů, tak je následně schopna řešit i úlohy, se kterými se během učení ještě přímo vůbec nesetkala. A takováto schopnost generalizace už představuje malý zárodek umělé inteligence.

4.4.3 Učení bez učitele

V tomto případě jsou nám známy vstupy, ale již ne požadovaný výstup. Sítě s tímto typem učení, mají většinou za úkol provádět tzv. shlukovou analýzu. To vlastně znamená rozdělení zadané množiny vstupů do konečného množství tříd. Vstupy z jedné třídy si jsou vzájemně podobné. Lze je tedy následně snadno podrobit další analýze, ať už ji bude provádět člověk nebo třeba další neuronová síť.

4.5 Příklady použití neuronových sítí

Uveďme si několik typických oblastí, ve kterých se neuronové sítě nejvíce používají:

- Rozpoznávání poruch strojů (např. diagnostika automobilů).
- Analýza textu (např. rozlišení nevyžádané pošty).
- Analýza hlasu (např. převod mluvené řeči do písemné podoby).
- Analýza obrazu (např. rozpoznávání tváří, diagnostika nádorů z rentgenových snímků, rozpoznání SPZ ze záběrů z kamery).
- Finanční sféra (např. predikce budoucího vývoje cen komodit, automatické obchodování s akciemi na burze, identifikace podezřelých obchodů, analýza rizikovosti klienta).
- Marketing (např. rozdělení zákazníků do skupin podle preferencí kvůli cílení reklamy).
- Samoobslužné mechanismy (např. automatické ovládání obráběcích strojů, autopilot).
- Asistenční aplikace (těm je věnována tato práce).

4.6 Způsob řešení problému v rámci této práce

V rámci této práce budeme k implementaci „mozku“ aplikace využívat serverový backend napsaný v jazyce PHP a využívající technologii MySQL. Implementace editační vzdálenosti bude fungovat na principu Levenshteinovy vzdálenosti.

5 Syntéza řeči

Technologie převodu řeči na text (text to speech, TTS) je posledním článkem v řetězci komunikace aplikace s uživatelem. Zajišťuje mluvenou odpověď (zpětnou vazbu).

Akustická syntéza řeči je proces či technika vytváření řečového signálu, tj. mluvené řeči. Počítačový systém používaný pro tento účel se nazývá řečový počítač nebo syntetizér řeči a lze jej implementovat do softwarových nebo hardwarových produktů.[38] Jejím cílem je vytvářet řeč v takové formě a kvalitě, aby co nejvěrněji kopírovala řečové charakteristiky lidí. Nejen samotný hlas a jeho kvalitu, ale i styl mluvení.

Tento proces se skládá ze sady speciálních modulů a algoritmů. Zahrnují zpracování textu (např. analýzu a normalizaci), převod textu do výslovnostní podoby (tj. fonetickou transkripci a generování průběhů prozodických vlastností řeči) a vlastní metodu vytváření řeči.

K vytváření řeči se v současnosti využívá velké množství různých přístupů. Dvě primární technologie vytvářející syntetické řečové průběhy jsou zřetězená syntéza a formantová syntéza. Každá technologie má silné a slabé stránky a zamýšlená použití syntetického systému obvykle určí, který přístup se použije.

5.1 Zřetězená syntéza

Zřetězená (konkatenační) syntéza je založena na zřetězení (nebo provázání) segmentů zaznamenané řeči. Obecně řečeno, zřetězená syntéza produkuje syntetizovanou řeč, která je nejpřirozenější. Rozdíly mezi přirozenými změnami řeči a povahou automatizovaných technik pro segmentaci průběhů však někdy vedou k slyšitelným závadám ve výstupu. Existují tři hlavní podtypy zřetězené syntézy.

5.1.1 Syntéza výběru jednotek

Tato metoda používá velké databáze zaznamenané řeči.

Během vytváření databáze je každý zaznamenaný projev rozdělen do několika nebo všech z následujících částí: jednotlivé fóny, difóny, polofóny, slabiky, morfémy, slova, fráze a věty.

Rozdělení na segmenty se obvykle provádí pomocí speciálně upraveného rozpoznávače řeči nastaveného na režim „vynuceného zarovnání“ s následnou ruční korekcí, za použití vizuálních reprezentací, jako je např. tvar vlny a spektrogram.[39] Index jednotek v databázi řeči je poté vytvořen na základě segmentace akustických parametrů, jako je základní frekvence (rozteč), doba trvání, pozice v slabice a sousední fóny. V době běhu je požadovaný cílový výrok vytvořen určením nejlepšího řetězce kandidátních jednotek z databáze. Tento proces je obvykle dosažen pomocí speciálně váženého rozhodovacího stromu.

Výběr jednotky poskytuje největší přirozenost, protože na zaznamenanou řeč aplikuje pouze malé množství digitálního zpracování signálu (DSP). DSP často dělá nahraný zvuk řeči méně přirozeným, ačkoli některé systémy používají malé množství zpracování signálu v bodě zřetězení k vyhlazení tvaru vlny. Výstup z nejlepších systémů výběru jednotek je často nerozeznatelný od skutečných lidských hlasů, zejména v kontextech, pro které byl systém

TTS vyladěn. Avšak maximální přirozenost obvykle vyžaduje, aby databáze s výběrem jednotek byly velmi velké, v některých systémech sahajících do gigabajtů zaznamenaných dat, což představuje desítky hodin řeči.[40] O algoritmech výběru jednotek je také známo, že vybírají segmenty z místa, což má za následek méně kvalitní syntézu (např. menší slova se stanou nejasnými), i když v databázi existuje lepší výběr.[41] V poslední době vědci navrhli různé automatizované metody detekce nepřirozených segmentů v jednotkových selekčních systémech syntézy řeči.[42]

5.1.2 Difónová syntéza

Tato syntéza používá minimální databázi řeči obsahující všechny difóny (přechody zvuku na zvuk) vyskytující se v jazyce. Počet difónů závisí na fonotaktice jazyka: například španělština má asi 800 difónů a němčina asi 2500. Při syntéze difónů je v databázi řeči obsažen pouze jeden příklad každého difónu. Za běhu je cílová prozodie věty superponována na tyto minimální jednotky pomocí technik digitálního zpracování signálu, jako je lineární prediktivní kódování, PSOLA[43] nebo MBROLA.[44] nebo novější techniky, jako je modifikace výšky tónu ve zdrojové doméně pomocí diskrétní kosinové transformace.[45] Difónová syntéza trpí zvukovými závadami zřetězené syntézy a roboticky znějící podstatou syntézy formantů a má jen málo výhod jednoho z přístupů jiných než malých rozměrů. Jeho použití jako takového v komerčních aplikacích klesá, ačkoli je používán ve výzkumu, protože tam je množství volně dostupných softwarových implementací. Prvním příkladem této syntézy je výukový robot, Leachim, který vynalezl Michael J. Freeman.[46] Leachim obsahoval informace týkající se učebních osnov a určité biografické informace o 40 studentech, kteří byli učeni.[47] Testování bylo provedeno ve čtvrté třídě v Bronxu v New Yorku.[48][49]

5.1.3 Syntéza specifická pro doménu

Syntéza specifická pro doménu zřetězuje předem zaznamenaná slova a fráze, aby se vytvořily úplné promluvy. Používá se v aplikacích, kde je množství textů, které systém vydá, je omezeno na konkrétní doménu, jako jsou oznámení o přepravním plánu nebo zprávy o počasí.[50] Tato technologie je velmi jednoduchá na implementaci a byla komerčně používána po dlouhou dobu v zařízeních, jako jsou mluvící hodinky a kalkulačky. Úroveň přirozenosti těchto systémů může být velmi vysoká, protože rozmanitost typů vět je omezená a úzce se shodují s prozódií a intonací původních nahrávek.

Protože tyto systémy jsou omezeny slovy a frázemi v jejich databázích, nejsou univerzální a mohou syntetizovat pouze kombinace slov a frází, s nimiž byly předprogramovány. Míchání slov v přirozeně mluveném jazyce však může stále způsobovat problémy, pokud se nezohlední mnoho variací. Například v nerotických dialektech angličtiny je slovo „r“ ve slovech jako „clear“ / ˈklɪə / obvykle vyslovováno pouze tehdy, když má následující slovo samohlásku jako své první písmeno (např. „Clear out“ je realizováno jako / ˌklɪəɹˈʌʊt /). Podobně ve francouzštině mnoho konečných souhlásek se stane najednou slyšitelnými, pokud za nimi následuje slovo, které začíná samohláskou, což je efekt nazývaný styk. Tuto alternaci nelze reprodukovat jednoduchým systémem zřetězení slov, který by vyžadoval další složitost, aby byla kontextově citlivá.

5.2 Formantová syntéza

Syntéza formantu nepoužívá vzorky lidské řeči za běhu. Místo toho je syntetizovaný řečový výstup vytvořen pomocí aditivní syntézy a akustického modelu (syntéza fyzikálního modelování).[51] Parametry, jako je základní frekvence, hlasitost a úroveň šumu, se v průběhu času mění, aby se vytvořil tvar umělé řeči. Tato metoda se někdy nazývá syntéza založená na pravidlech; mnoho zřetězených systémů však také obsahuje komponenty založené na pravidlech. Mnoho systémů založených na technologii syntézy formantu vytváří umělou řeč, roboticky znějící, která by se nikdy neměla zaměnit za lidskou řeč. Maximální přirozenost však není vždy cílem systému syntézy řeči a systémy syntézy formantu mají výhody oproti systémům zřetězeným. Formálně syntetizovaná řeč může být spolehlivě srozumitelná, dokonce i při velmi vysokých rychlostech, a umožňuje vyhnout se akustickým závadám, které běžně trápí spojovací systémy. Vysokorychlostní syntetizovaná řeč je používána zrakově postiženými k rychlé navigaci počítačů pomocí čtečky obrazovky. Syntetizátory formantů jsou obvykle menšími programy než zřetězené systémy, protože nemají databázi vzorků řeči. Mohou být proto použity ve vestavěných systémech, kde je paměť a mikroprocesorový výkon zvláště omezený. Protože systémy založené na formantu mají úplnou kontrolu nad všemi aspekty výstupní řeči, může být vydána široká škála prozódií a intonací, které přinášejí nejen otázky a výroky, ale i různé emoce a tóny hlasu.

Příklady nereálného, ale vysoce přesného řízení intonace v syntéze formantů zahrnují práci provedenou v pozdních sedmdesátých letech pro hračku Texas Instruments Speak & Spell a na počátku osmdesátých let arkádové automaty Sega[52] a v mnoha Atari, Inc. arkádové hry[53] využívající čipy TMS5220 LPC. Vytváření správné intonace pro tyto projekty bylo pečlivé a výsledky ale ještě musí být sladěny s rozhraními převodu textu na řeč v reálném čase.[54]

5.3 Kvalita a užití hlasových syntetizátorů

Kvalita syntetizátoru řeči se posuzuje podle jeho podobnosti s lidským hlasem a jeho srozumitelnosti.

Srozumitelný program převodu textu na řeč umožňuje lidem se zrakovým postižením nechat si přečíst nějaký text nebo obsah na obrazovce. U lidí s vadou řeči tento systém dokáže částečně nahradit jejich řeč (příkladem takto handicapovaného člověka může být jeden z nejznámějších vědců všech dob profesor Stephen Hawking).

Mnoho počítačových operačních systémů zahrnuje syntezátory řeči od počátku 90. let.

6 Vlastní implementace hlasového agenta

V této části popíšeme samotnou implementaci „inteligentní“ aplikace pro platformu iOS.

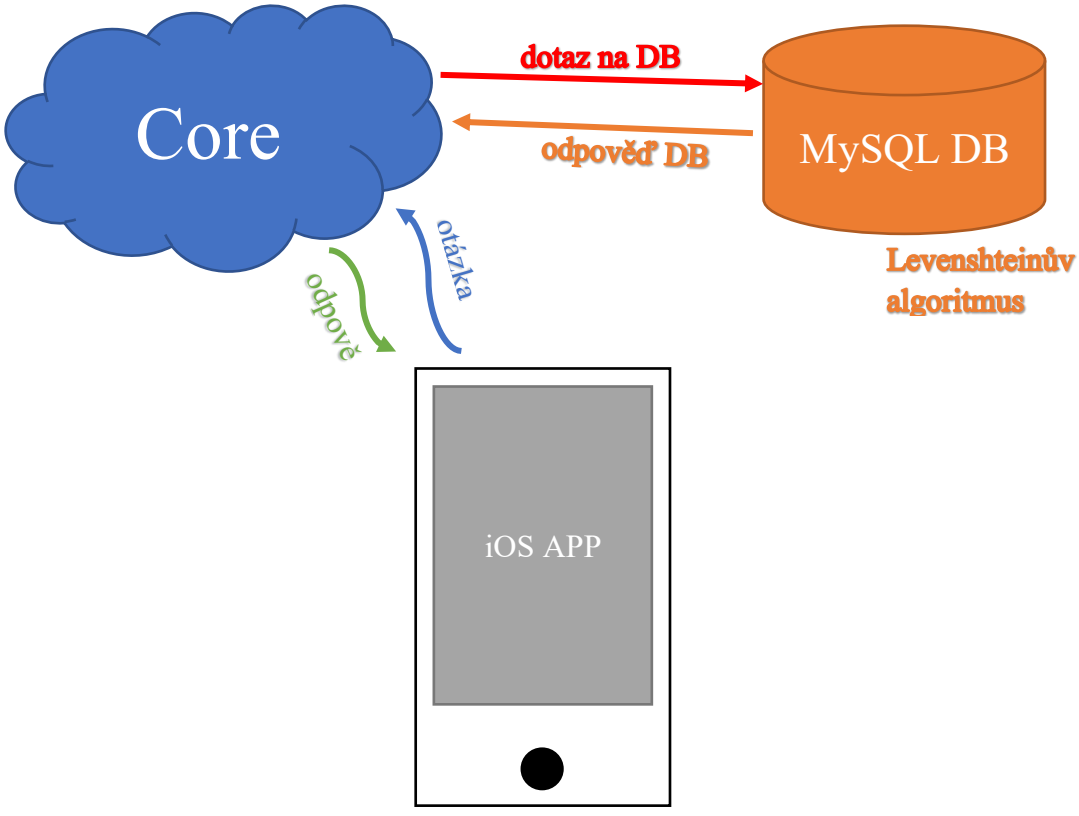
6.1 Návrh architektury celého systému

Celý systém bude postaven na multiplatformní bázi, tedy veškeré majoritní funkce poběží na PHP serveru (Core) s přístupem do MySQL databáze MariaDB. Tím bude dosaženo možnosti jednoduchého rozšíření tohoto softwaru i na jiné platformy, než je iOS. Zároveň získáme možnost kdykoliv udělat úpravu „mozku“ aplikace, a to bez nutnosti aktualizace samotné mobilní aplikace. Jak jsme již uvedli, v rámci této implementace použijeme databázi znalostí (čítající přes 2000 frází) ze zrušeného projektu ClaraSys pro OS Android.

Samotná mobilní aplikace bude vyvinutá v programovacím jazyce Swift. Bude fungovat na jednoduchém principu: získáme hlasový vstup, který převedeme na text. Ten následně pošleme serveru a jako odpověď dostaneme opět text, který pouze převedeme na řeč pomocí hlasového syntezátoru.

Abychom funkcionalitu takové aplikace ještě o něco rozšířili, implementujeme mnou vyvinutou knihovnu ApplicationAPI, která nám umožní měnit kód aplikace za běhu. Na tuto knihovnu se blíže zaměříme později.

V průběhu implementace Levenshteinova algoritmu jsem zjistil, že tento postup je hodně výpočetně náročný, a potřebuje zřejmě větší výpočetní výkon. Proto jsem se rozhodl pro srovnání implementovat ještě druhý algoritmus, který funguje totožně, ale nevyžaduje takový hardwarový výkon. Porovnání rychlostí odpovědí u obou algoritmů se budeme věnovat v následující kapitole.



Obrázek 13 Návrh aplikace (schéma)

6.2 Rozpoznání řeči

Pro technologii rozpoznání řeči a její převodu na text jsem v rámci tohoto projektu zvolil hotové řešení od společnosti Apple – API SpeechRecognizer. Očekává se, že tohle řešení by právě na iOS zařízeních mělo fungovat nejlépe a mělo mít snadnou implementaci.

Knihovnu SpeechRecognizer není třeba do projektu nijak přidávat, protože je automaticky součástí SDK (Software Development Kit). Stačí nám tedy pouze přidat import statement:

```
import Speech
import AVFoundation
```

První import nám naimportuje do projektu a jeho namespace API SpeechRecognizer, druhý nám zajistí napojení na knihovnu pro práci se zvukem (potřebujeme hlas nahrát).

Poté deklarujeme potřebné třídní proměnné:

```
private var speechRecognizer = SFSpeechRecognizer(locale: Locale.init(identifier: "cs-CZ"))
private var recognitionRequest: SFSpeechAudioBufferRecognitionRequest?
private var recognitionTask: SFSpeechRecognitionTask?
private var audioEngine = AVAudioEngine()
```

Ve funkci *viewDidLoad()*, která se spustí při „otevření aplikace“ (ve skutečnosti při otevření daného ViewControlleru), přidáme následující řádek, který nám umožní přijímat výsledky rozpoznání řeči v této třídě (ViewControlleru):

```
speechRecognizer?.delegate = self as? SFSpeechRecognizerDelegate
```

Dále je třeba v téže funkci ještě přidat kód, který nám získá oprávnění používat mikrofon:

```
SFSpeechRecognizer.requestAuthorization{ (authStatus) in
            var isButtonEnabled = false

            switch (authStatus){
            case .authorized:
                isButtonEnabled = true

            case .denied:
                isButtonEnabled = false
                print("User denied access.")

            case . restricted:
                isButtonEnabled = false
                print("Restricted.")

            case .notDetermined:
                isButtonEnabled = false
                print("Not yet authorized.")
            }

            OperationQueue.main.addOperation() {
                self.button.isEnabled = isButtonEnabled
                if isButtonEnabled {
                    self.button.backgroundColor = UIColor.blue
                }else{
                    self.button.backgroundColor = UIColor.lightGray
                }
            }
        }
```

Po kliknutí na tlačítko „Začít poslouchat“ spustíme funkci *startRecording()*:

```
func startRecording() {

        if recognitionTask != nil {
            recognitionTask?.cancel()
            recognitionTask = nil
        }

        do{
            try AVAudioSession.sharedInstance().setActive(false, options:
.notifyOthersOnDeactivation)
            try
AVAudioSession.sharedInstance().setCategory(AVAudioSession.Category.playAndRecord, mode:
.default, options: .defaultToSpeaker)
            try AVAudioSession.sharedInstance().setMode(AVAudioSession.Mode.measurement)
            try AVAudioSession.sharedInstance().setActive(true, options:
.notifyOthersOnDeactivation)
        }catch {
            print("AudioSession selhalo.")
        }

        recognitionRequest = SFSpeechAudioBufferRecognitionRequest()

        let inputNode = audioEngine.inputNode

        guard let recognitionRequest = recognitionRequest else {
            fatalError("Objekt SFSpeechAudioBufferRecognitionRequest nemohl být načten.")
        }

        recognitionRequest.shouldReportPartialResults = true

        recognitionTask = speechRecognizer?.recognitionTask(with: recognitionRequest,
resultHandler: { (result, error) in

            var isFinal = false

            if result != nil {

                self.txtMe.text = result?.bestTranscription.formattedString

                self.said = result?.bestTranscription.formattedString ?? ""

                self.txtMe.centerVertically()
                self.txtClara.centerVertically()

                print(self.said + "/n")

                isFinal = (result?.isFinal)!
            }

            if error != nil || isFinal {
                self.audioEngine.stop()
                inputNode.removeTap(onBus: 0)

                self.recognitionRequest = nil
                self.recognitionTask = nil

                self.button.isEnabled = true
                self.button.backgroundColor = UIColor.blue
            }
        })

        let recordingFormat = inputNode.outputFormat(forBus: 0)
        inputNode.installTap(onBus: 0, bufferSize: 1024, format: recordingFormat) { (buffer,
when) in
            self.recognitionRequest?.append(buffer)
        }

        audioEngine.prepare()

        do {
            try audioEngine.start()
        } catch {
            print("AudioEngine selhal..")
        }

        txtMe.text = "Poslouchám..."

        txtMe.centerVertically()
```

Funkce nahraje zvuk a odešle ho na servery společnosti Apple. Následně v ní nastavíme handler, který nám umožní zpracovat výsledný text, který obdržíme po dokončení procesu. Apple servery nečekají, až bude dokončena celá věta či spojení, průběžně odesílají každé jednotlivé slovo k rozpoznání a jakmile provedou analýzu, volají náš handler. Ten tak průběžně dostává jednotlivá vyřčená slova.

Ve funkci používáme proměnnou *said* – ta slouží právě k průběžnému uchovávání výsledného textu. Dále zde máme 2 textová pole – jedno pro zobrazení vyřčeného textu (*txtMe*), druhé pro zobrazení odpovědi z naší aplikace (*txtClara*, protože jsem tuto „inteligentní“ bytost pojmenoval Klára).

API na rozpoznání řeči tak funguje kontinuálně a průběžně nám vypisuje do textového pole *txtMe* jednotlivá vyřčená slova (která řetězí za sebou). Aby aplikace zjistila, že jsme již vyslovili vše, co jsme chtěli, musíme jí to dát vědět. K tomu nám slouží tlačítko „Přestat poslouchat“. Po kliknutí na něho se nám spustí tato funkce:

```
func stopListening() {
       audioEngine.stop()
       recognitionRequest?.endAudio()
       answer()
}
```

V ten okamžik máme v naší proměnné *said* uložen veškerý text, který uživatel nahlas vyslovil a Apple servery ho správně rozpoznaly.

Funkce obsahuje i další kód týkající se GUI (grafického uživatelského prostředí). Tento kód zde však pro větší přehlednost vynechávám.

6.3 Komunikace se serverem (Core)

Funkce *stopListening()* na svém konci zavolá metodu *answer()* – ta má na starosti odeslání textu na server:

```
func answer(){
        var url_address =
"https://core.clarasys.cz/mozek/index.php?auth=dcuihuhfduh566hgjhghjk6IGBUZGGut7GHI7Z&query=" +
said;
        url_address = url_address.addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed)
?? ""
        print(url_address)
        let url = URL(string: url_address)!

        let task = URLSession.shared.dataTask(with: url) {(data, response, error) in
            guard let data = data else { return }

            var answer = String(data: data, encoding: .utf8)!

            OperationQueue.main.addOperation() {
                if(answer.contains("&;&")){
                    let array = answer.components(separatedBy: "&;&")

                    answer = array[0]

                    ApplicationAPI().commit(cmd: array[1])
                }

                self.txtClara.text = answer
                self.speak(text: answer)

                self.txtMe.centerVertically()
                self.txtClara.centerVertically()
            }
        }

        task.resume()
```

Odesílání textu na server se provádí metodou GET, text se odesílá v parametru *query*. Aby na server nemohl zadávat požadavky kdokoliv, obsahuje PHP script i primitivní úroveň zabezpečení pomocí tajného klíče v GET parametru *auth*.

Po získání odpovědi ze serveru v textové podobě se provede nejprve kontrola, zdali odpověď obsahuje sekvenci znaků *&;&*. Je to z toho důvodu, že tato sekvence rozděluje textovou odpověď „Kláry“ od případných příkazů v jazyce ApplicationAPI. Tomu se budeme dále věnovat později v tomu určené kapitole.

Zásadní je zde pro nás zavolání metody *speak()*, která odešle jako svůj parametr získanou odpověď ze serveru a pošle ji k vyřčení hlasovému syntetizátoru. Tomu se budeme věnovat v následující kapitole.

6.4 Hlasový syntetizátor

Pro tento projekt opět zvolíme řešení od společnosti Apple, které je již součástí systému. Po implementaci tohoto řešení ponecháme aplikaci možnost odpovědět uživateli „přirozenou“ formou.

Výhodná je skutečnost, že všechno potřebné již máme implementováno. Jediné, co nám ještě chybí, je samotná funkce *speak()*:

```
func speak(text: String){
        do {
            try AVAudioSession.sharedInstance().setActive(false, options:
.notifyOthersOnDeactivation)
            try
AVAudioSession.sharedInstance().setCategory(AVAudioSession.Category.playAndRecord, mode:
.default, options: .defaultToSpeaker)
            try AVAudioSession.sharedInstance().setActive(true, options:
.notifyOthersOnDeactivation)
        } catch {
            print("AudioSession properties nebyly načteny.")
        }

        // Line 1. Create an instance of AVSpeechSynthesizer.
        let speechSynthesizer = AVSpeechSynthesizer()
        // Line 2. Create an instance of AVSpeechUtterance and pass in a String to be spoken.
        let speechUtterance: AVSpeechUtterance = AVSpeechUtterance(string: text)
        //Line 3. Specify the speech utterance rate. 1 = speaking extremely the higher the
values the slower speech patterns. The default rate, AVSpeechUtteranceDefaultSpeechRate is 0.5
        speechUtterance.rate = AVSpeechUtteranceMaximumSpeechRate / 2.0
        // Line 4. Specify the voice. It is explicitly set to English here, but it will use the
device default if not specified.
        speechUtterance.voice = AVSpeechSynthesisVoice(language: "cs-CZ")
        // Line 5. Pass in the urrerance to the synthesizer to actually speak.
```

Tato funkce se kompletně postará o náš hlasový výstup.

6.5 ApplicationAPI

U aplikace nám zbývá poslední neprobraná záležitost – ApplicationAPI.

Jedná se o knihovnu, kterou jsem vyvinul pod OS Android v průběhu mé působnosti v rámci projektu ClaraSys. Potřebovali jsme průběžně přidávat funkce aplikace (budík, připomínky, zobrazení obrázků, webů, videí, ovládání nastavení, hlasitostí a zvukový profilů apod.). Nepřicházelo však v úvahu, abychom s každou novou funkcí vydávali novou verzi aplikace.

Podle našich zkušeností si jen asi 30 % uživatelů do měsíce stáhlo aktualizaci aplikace. Tomu jsme se chtěli vyhnout, přáli jsme si, aby měli všichni uživatelé nové funkce okamžitě.

Ještě větší uplatnění to mělo v případě nutnosti odstranění nějakých chyb. Uživatelé si museli aplikaci aktualizovat, aby se jim chyba opravila. (Problémem bylo i zdržení s exportem aplikace a následným zdlouhavým nahráním do Google Play, nehledě na schvalovací proces každé aktualizace).

Vymyslel jsem tedy takový vlastní pseudo-programovací jazyk, jehož příkazy se vždy posílaly s odpovědí ze serveru právě za oddělovačem *&;&*. Tyto příkazy si následně aplikace rozkouskovala a jeden po druhém je vykonala. Aplikace fungovala na principu zachycování názvů příkazů a extrahování jejich parametrů. Následně jsem jednoduchým switch statementem ověřil, o který příkaz se jedná. Ten jsem následně vykonal s danými parametry.

Uvedu zde příklad postupu řešení. Poslouží nám k němu příkaz na rozsvícení (zhasnutí) zabudované baterky v telefonu (blesk diody):

flashlight();

Tento příkaz nám buďto aktivuje nebo deaktivuje svítilnu (podle aktuálního stavu rozsvícení).

Uveďme si ještě jeden, složitější příkaz – s parametrem:

showWeb(url);

Tento příkaz nám otevře webovou stránku s URL adresou uvedenou jako parametr.

Samozřejmě existují i složitější příkazy, jako například *setRingerMode(silent);* pro nastavení tichého zvukového profilu, nebo *setMediaVolume(max);* na zvýšení hlasitosti médií na maximum. Těch příkazů je velké množství a dají se také řetězit:

setMediaVolume(5);showWeb(https://youtube.com/zsgfsdfjksdlf);

Důležité je zmínit, že takové široké možnosti využití tohoto API je možné pouze v rámci platformy Android. iOS je totiž velmi uzavřený systém a 90 % všech příkazů by na úrovni aplikace 3. strany nešlo vůbec zrealizovat. Proto jsem v rámci této práce vytvořil takovou slabší modifikaci mého API pro platformu iOS, spíše k demonstrativním účelům.

Implementoval jsem zde tedy 3 funkce: *flashlight()*, *showWeb()* a *playYT()*. Níže jsou ony funkce zastupující tyto příkazy na úrovni knihovny:

```
func flashlight() {
        guard let device = AVCaptureDevice.default(for: AVMediaType.video) else { return }
        guard device.hasTorch else { return }

        do {
            try device.lockForConfiguration()

            if (device.torchMode == AVCaptureDevice.TorchMode.on) {
                device.torchMode = AVCaptureDevice.TorchMode.off
            } else {
                do {
                    try device.setTorchModeOn(level: 1.0)
                } catch {
                    print(error)
                }
            }

            device.unlockForConfiguration()
        } catch {
            print(error)
        }
    }


    func showWeb(_ url: String){
        guard let url = URL(string: url) else { return }
        UIApplication.shared.open(url)
    }

    func playYT(_ url: String){
        showWeb("https://www.youtube.com/watch?v=" + url)
    }
```

Přestože jsou na platformě iOS velká omezení, i zde má tato knihovna smysl. Nedají se sice takto „automatizovat“ systémová nastavení, ale dá se nalézt uplatnění v jiných oblastech. Největším problémem je zde asi neschopnost odesílat SMS zprávy pomocí hlasových povelů. To nám takto uzavřený systém nedovolí.

6.6 Implementace PHP serveru (Core)

Zde jsem opět pro účely výkladu celou implementaci o něco zjednodušil.

Když přijde dotaz na server, nejprve se zkusí zjistit, zda daný dotaz neobsahuje nějaká klíčová slova – to by znamenalo, že se nejedná o pouhou komunikační frázi, ale o funkční dotaz. Příkladem takového funkčního dotazu může být jeden z následujících dotazů:

Kolik je hodin?
Co je za den?
Kdo má zítra svátek?
Mají dnes v Kauflandu v Trutnově otevřeno?
Najdi mi nějaký hezký vtip
Co je to automobil?
Kdo je Karel Gott?
Pusť mi na YouTube Pata a Mata
Vyhledej restaurace v okolí Trutnova
Jaké je počasí v Olomouci?
Jaké bude počasí v Hradci Králové?

Kolik je 2 krát 2?
Kolik je 16 na druhou?
V rámci zjednodušení jsem nechal pouze výše zmíněné funkce. U těch jednodušších stačí základní algoritmus v PHP (*Kolik je hodin?* nebo *Kolik je 16 na druhou?*), u těch složitějších odpověď získáme přes API třetí strany:

Kdo nebo co je? – Wikipedia API
Kdy mají kde otevřeno – Google API
Počasí – OpenWeatherMap.org

Všechny zmíněné služby jsou založené na REST API. Odešle se dotaz v parametru URL a server třetí strany nám zašle odpověď v JSON formátu. Zdrojový kód těchto funkcí je velmi dlouhý, ale pro ukázku níže přikládám část kódu pro získání dnešního počasí na základě zadaného města:

```
$url="http://api.openweathermap.org/data/2.5/forecast/daily?q=".$city.",".$country."&units=metric&cnt=1
&lang=cz&appid=686eefc78163f04a88bbf08351c29fdc";
$json=file_get_contents($url);
$data=json_decode($json,true);

$we = $data['list'][0]['weather'][0]['description'];

$wtt = $data['list'][0]['temp']['min'];
$wttm = $data['list'][0]['temp']['max'];

$wtmin = floor($wtt);
$wtmax = floor($wttm);

$wt = ($wtmin + $wtmax) / 2;

$vystup = 'Zítra bude ve městě ' . $city . ' ' . $we . ' a průměrně bude ' . $wt . ' stupňů.';
```

Když dotaz z aplikace na náš PHP server (Core) neobsahuje žádné klíčové slovo, přejde se na zvolení nejvhodnější odpovědi z DB. K výpočtu nejvhodnější odpovědi využijeme právě dříve popsaný Levenshteinův algoritmus.

Nejprve vytvoříme funkci levenshtein() na MySQL serveru. Funkce vypadá následovně:

```
CREATE FUNCTION levenshtein( s1 VARCHAR(255), s2 VARCHAR(255) )
  RETURNS INT
  DETERMINISTIC
  BEGIN
    DECLARE s1_len, s2_len, i, j, c, c_temp, cost INT;
    DECLARE s1_char CHAR;
    -- max strlen=255
    DECLARE cv0, cv1 VARBINARY(256);
    SET s1_len = CHAR_LENGTH(s1), s2_len = CHAR_LENGTH(s2), cv1 = 0x00, j = 1, i = 1, c = 0;
    IF s1 = s2 THEN
      RETURN 0;
    ELSEIF s1_len = 0 THEN
      RETURN s2_len;
    ELSEIF s2_len = 0 THEN
      RETURN s1_len;
    ELSE
      WHILE j <= s2_len DO
        SET cv1 = CONCAT(cv1, UNHEX(HEX(j))), j = j + 1;
      END WHILE;
      WHILE i <= s1_len DO
        SET s1_char = SUBSTRING(s1, i, 1), c = i, cv0 = UNHEX(HEX(i)), j = 1;
        WHILE j <= s2_len DO
          SET c = c + 1;
          IF s1_char = SUBSTRING(s2, j, 1) THEN
            SET cost = 0; ELSE SET cost = 1;
          END IF;
          SET c_temp = CONV(HEX(SUBSTRING(cv1, j, 1)), 16, 10) + cost;
          IF c > c_temp THEN SET c = c_temp; END IF;
            SET c_temp = CONV(HEX(SUBSTRING(cv1, j+1, 1)), 16, 10) + 1;
            IF c > c_temp THEN
              SET c = c_temp;
            END IF;
            SET cv0 = CONCAT(cv0, UNHEX(HEX(c))), j = j + 1;
        END WHILE;
        SET cv1 = cv0, i = i + 1;
      END WHILE;
    END IF;
    RETURN c;
  END;
```

Následně implementujeme dotaz na MySQL server v jazyce PHP. Než to však učiníme, zaměříme se ještě na strukturu databáze známých frází.

Struktura vypadá následovně:

Každý řádek tabulky (s názvem „core“) obsahuje (mimo jiné pomocné sloupce) 3 sloupce:

1) *id*: Jednoznačné ID fráze
2) *question*: Otázka na danou frázi
3) *answer*: Odpověď na danou frázi

Tedy známe otázku (*question*; vstup od uživatele) a musíme v databázi najít tu nejpodobnější. Tím zjistíme její *id*, které následně použijeme k získání relevantní odpovědi (*answer*).

Příklad takového řádku:

id	***question***	***answer***
1234	Ahoj	Ahoj, potřebuješ s něčím pomoct?

V PHP tento dotaz na databázi provedeme následovně:

```
$mysqli->set_charset("utf8");

/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}

$query = "SELECT answer FROM core
ORDER BY levenshtein('$said',`question`)
LIMIT 1";

$result = $mysqli->query($query);

$row = $result->fetch_array(MYSQLI_ASSOC);

$vystup = $row['answer'];
```

Do proměnné *$vystup* se nám uloží nejvíce relevantní odpověď, tu na konci vypíšeme na stránku, kde si ji stáhne mobilní aplikace, a další postup už známe.

6.7 Výkon & další způsob řešení

Naše aplikace je funkční, ale samotný Levenshteinův algoritmus trvá našemu MySQL serveru 6 000 milisekund (tj. 6 sekund). Aplikace tak vysloví odpověď za přibližně 7 sekund od kliknutí na tlačítko „Přestat poslouchat“. To není úplně dobrá zpráva. Při každém dotazu se procesor na serveru dostane na 100 % svého vytížení, nestačí nám tedy výkon. Proto ukážeme ještě jedno – mnohem rychlejší řešení:

Místo Levenshteinova algoritmu můžeme pro výpočet podobnosti textových řetězců použít funkci z PHP. Musíme nejdříve z MySQL databázového serveru vytáhnout všechny fráze, pak je v PHP porovnat (na základě procentuální podobnosti), a nakonec vypsat tu nejrelevantnější. Můžete předpokládat, že funkce na úrovni MySQL serveru je mnohem rychlejší, než vytažení všech záznamů a jejich porovnání v PHP. Realita je ovšem jiná. Takto položený dotaz zabere aplikaci 1 000 milisekund (tzn. jen jednu sekundu) od stisknutí tlačítka „Přestat poslouchat“.

Ke zjištění procentuální podobnosti využijeme PHP funkci:

similar_text($line['question'], $stringToFind, $percentage);

Dále proto uvádíme výslednou PHP funkci:

```
function porovnat($stringToFind){
        again:
        //Pomocné buffery
        $maximum = 0;
        $maximumString = "";
        //Projití všemi záznami

        $alines = Db::queryAll('
                SELECT *
                FROM core
                ORDER BY id
                ');

        foreach ($alines as $line) {

            //Porovnání, vrácí hodnotu v proměnné $percentage
            similar_text($line['question'], $stringToFind, $percentage);

            //Pokud je % větší než maximum, tak je maximum procento a maximumString je tento řádek
            if($percentage > $maximum)
            {

                $maximum = $percentage;
                $maximumString = $line['id'];
            }

        }
        // Vypiš nejpodobnější string
        if($maximum <= 50){
            return 4229;
        } else {
            return $maximumString;
        }
    }
```

Tento kód nám vytáhne z DB nejvhodnější frázi.

V případě, že je procentuální podobnost k jiné frázi nižší než 50 %, vypíšeme raději frázi s *id 4229*, která obsahuje univerzální odpověď.

Vytvořili jsme tedy duplikát naší aplikace (i PHP serveru), který funguje na tomto druhém algoritmu. V další kapitole tyto dva algoritmy porovnáme.

7 Testování aplikace

Tato kapitola je věnována testování naší aplikace.

Budeme spolu porovnávat 2 podobné aplikace, které pracovně budeme označovat jako ClaraSys-LV (s implementací Levenshteinovy funkce) a ClaraSys (bez Levenhteinovy funkce).

Jak už jsem psal v předchozí kapitole, doba reakce se u těchto dvou jinak totožných verzí aplikace velmi liší.

ClaraSys-LV…… doba reakce 7 vteřin
ClaraSys…….. doba reakce 1 vteřina

Poznámka: Oba dva testy jsou prováděny na stejném iPhonu i PHP serveru (Core). K testu využívám iPhone 8 Plus a VPS server s OS Windows Server 2016 DataCenter. Hardware serveru: sdílený procesor řady Intel Core i7, 4GB RAM.

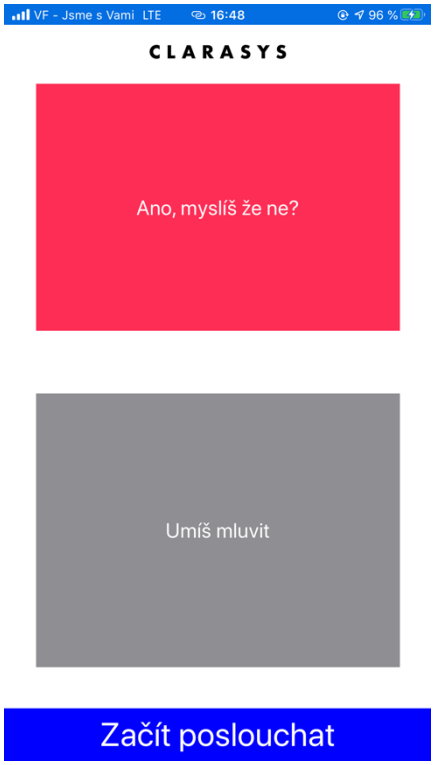
Při dotazu aplikace ClaraSys-LV je proces PHP serveru vytížen na 100 %. U dotazu aplikace ClaraSys je vytížení procesoru cca 20 %. Lze tedy s jistotou tvrdit, že ClaraSys má mnohem menší nároky na hardwarový výkon než ClaraSys-LV.

Nyní se zaměříme na kvalitu a přesnost odpovědí.

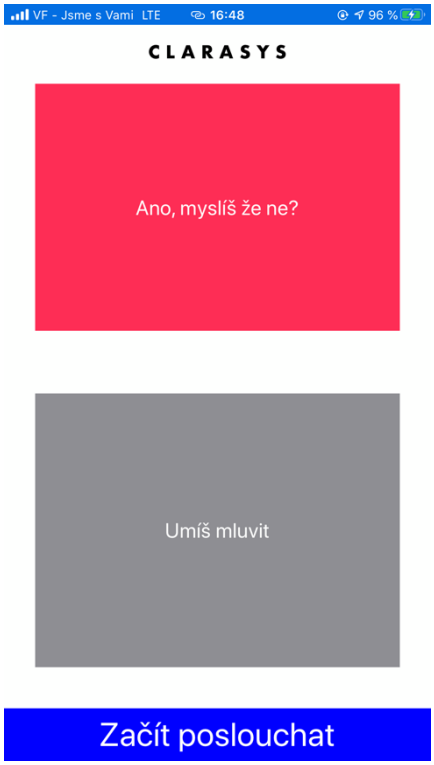
Tytéž otázky jsem položil oběma aplikacím. Následuje srovnání – nalevo je vždy ClaraSys-LV, napravo ClaraSys.

Obrázek 15 ClaraSys-LV Test č. 1

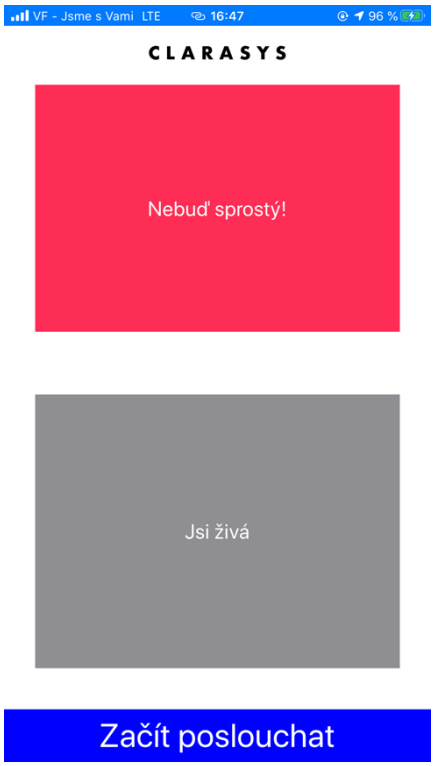
Obrázek 14 ClaraSys Test č. 1



Obrázek 17 ClaraSys-LV Test č. 2

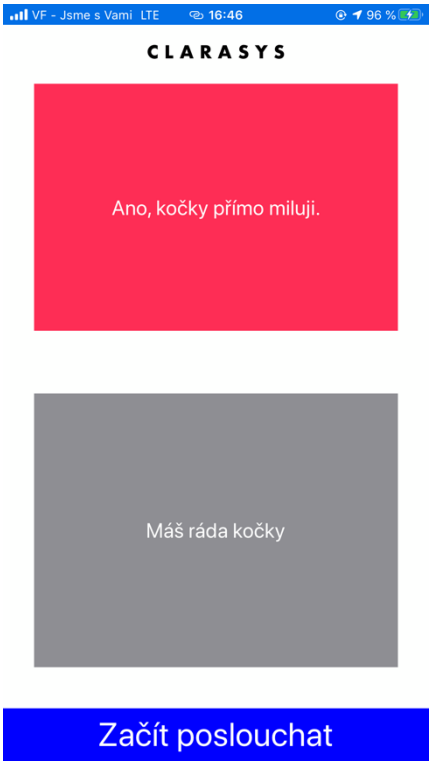


Obrázek 16 ClaraSys Test č. 2

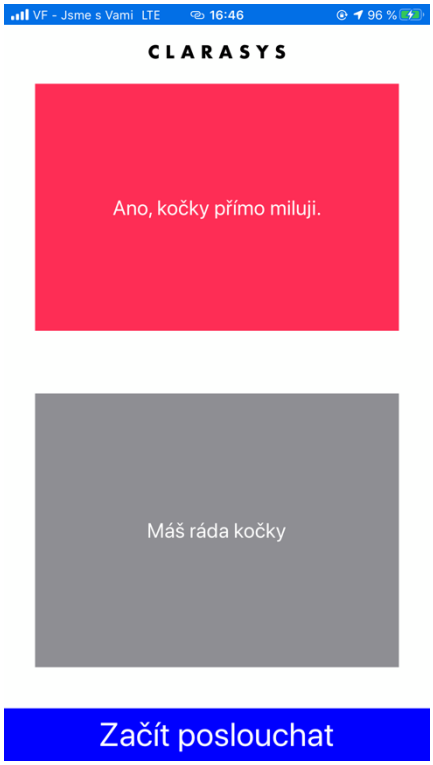


Obrázek 19 ClaraSys-LV Test č. 3 - CHYBA

Obrázek 18 ClaraSys Test č. 3



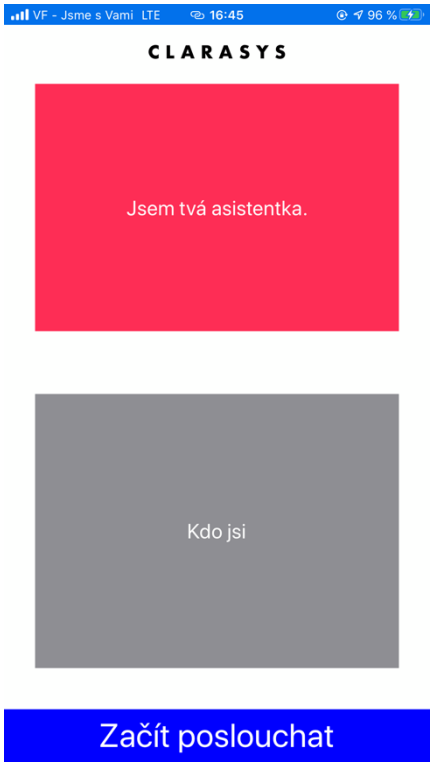
Obrázek 20 ClaraSys-LV Test č. 4



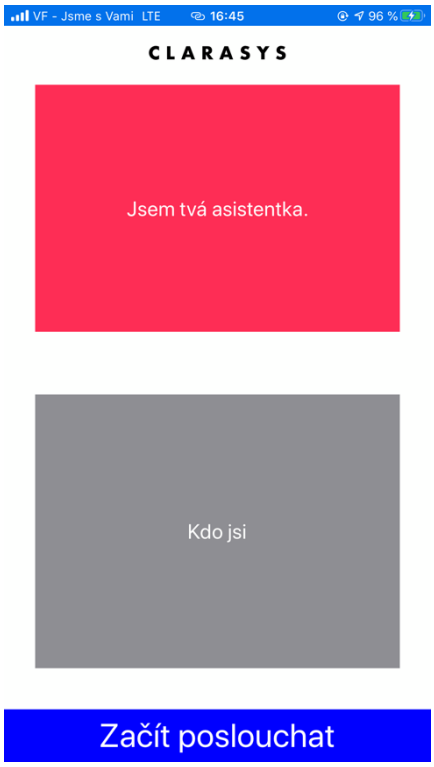
Obrázek 21 ClaraSys Test č. 4

Obrázek 23 ClaraSys-LV Test č. 5

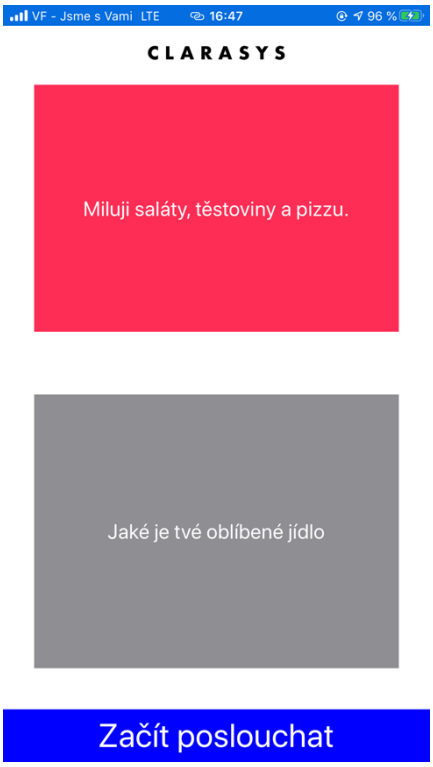
Obrázek 22 ClaraSys Test č. 5



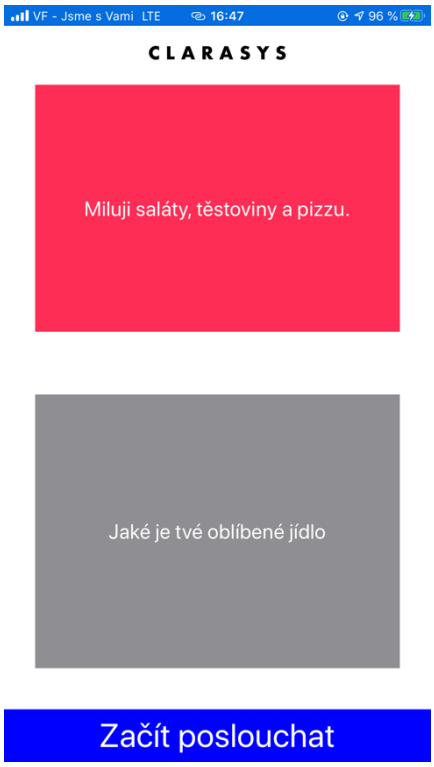
Obrázek 25 ClaraSys-LV Test č. 6



Obrázek 24 ClaraSys Test č. 6



Obrázek 27 ClaraSys-LV Test č. 7



Obrázek 26 ClaraSys Test č. 7

Obrázek 29 ClaraSys-LV Test č. 8

Obrázek 28 ClaraSys Test č. 8

Obrázek 31 ClaraSys-LV Test č. 9

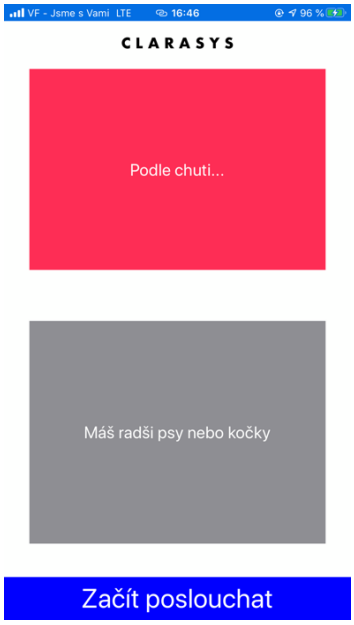
Obrázek 30 ClaraSys Test č. 9

Obrázek 33 ClaraSys-LV Test č. 10

Obrázek 32 ClaraSys Test č. 10

Jak můžeme vidět, téměř ve všech testovaných případech (kromě jedné, Test č. 3) odpověděly obě aplikace totožně. Přestože ClaraSys-LV trvá odpověď o celých 6 sekund déle! Navíc v Testu č. 3 měla aplikace ClaraSys-LV méně relevantní (dokonce až nesmyslnou) odpověď než aplikace ClaraSys.

Metoda použitá v aplikaci ClaraSys je nejen rychlejší, ale i lepší v relevantnosti odpovědí. Tento závěr byl překvapivý nejen pro autora práce.



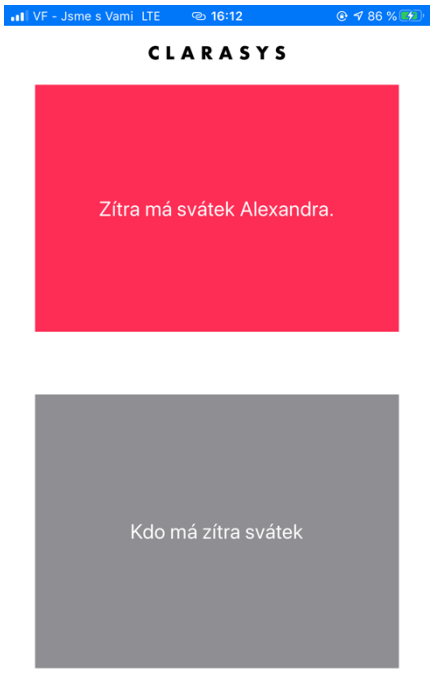
Ještě přiložíme jednu ukázku, která dopadla u obou aplikací stejně, a přestože dává odpověď smysl, je to nechtěná nahodilá asociace. Může však čtenáře této práce alespoň trochu pobavit.

Obrázek 34 Nechtěná nahodilá (avšak humorná) asociace

Na úplný závěr ještě demonstrujeme funkce aplikace, které jsou odchytávány ještě před prováděním frázových algoritmů.

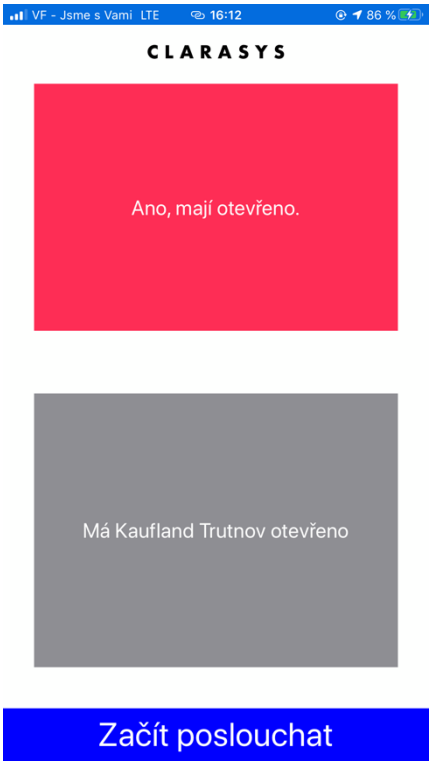
Obrázek 36 Počítání

Obrázek 35 Napojení na Wikipedii



Obrázek 38 Zjištění svátků (aplikace má v sobě databázi jmen)

Obrázek 37 Vtipy z internetu (náhodně vybrané a nahrané do databáze)



Obrázek 40 Napojení na Google API (firmy Google)

Obrázek 39 Napojení na OpenWeatherMap.org

8 Závěr, shrnutí a možné pokračování práce

V práci jsme se věnovali tématice „inteligentních“ mobilních hlasových asistentů a jejich nejznámějším příkladům realizace. Zaměřili jsme se na unikátní funkce a schopnosti každého z nich.

Vysvětlili jsme si obecné schéma architektury těchto asistentů. Následně jsme se zaměřili na každý jednotlivý prvek tohoto schématu: rozpoznání řeči, „inteligentní mozek“ celého systému a hlasový syntetizátor. Každý bod jsme detailně popsali a vysvětlili různé způsoby přístupu k doprovodným problémům spojeným s realizací.

Praktická část práce je věnována implementaci vlastního „inteligentního“ mobilního hlasového asistentka, resp. asistentky – Kláry. Za platformy jsme zvolili Apple iPhone (s operačním systémem iOS) a PHP server (Windows Server VPS) s MySQL databází (na tomtéž serveru).

Pro rozpoznání řeči i hlasový syntetizátor jsme využili hotová řešení od společnosti Apple.

Pro podporu „inteligentních“ rysů aplikace jsme zvolili 2 přístupy:

Levenshteinův algoritmus
funkci *similar_text()* z PHP

Nakonec jsme provedli testování těchto 2 variant v rámci stejné aplikace.

Jednoznačně nám jako efektivnější a rychlejší implementace vyšla druhá varianta s funkcí *simple_text()*. Navíc byly odpovědi v této variantě více relevantní.

Výsledek byl překvapivý pro autora práce, ale jistě také čtenáře.

Obecně považujeme implementaci za úspěšnou, ve většině testovaných případů by aplikace prošla úspěšně i Turingovým testem.

Zvolené téma by autor práce rád rozvedl do vyšší úrovně v rámci svojí diplomové práce na nadcházejícím magisterském studiu. Rád by místo vzdálenostních algoritmů a podobnosti frází implementoval neuronovou síť, a vytvořil inteligentní aplikaci se schopností učit se. Navíc by se rád soustředil na nějaké konkrétní zaměření takto inteligentního systému, v podstatě na způsob expertního systému. Takové řešení by mohlo mít cenné uplatnění v nějakém (složitějším) oboru.

Autor této práce je přesvědčen, že toto téma má velkou budoucnost a bude hrát významnou roli v dalším pokroku informačních technologií. Zejména lze předpokládat rozvoj těchto hlasových asistentů v kontextu chytrých domácností.

SEZNAM POUŽITÝCH ZDROJŮ

[1] Oficiální stránka Google Asistenta, https://assistant.google.com
[2] Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone". Google AI Blog. Retrieved October 15, 2018.
[3] Nieva, Richard (May 9, 2018). "Google Assistant's one step closer to passing the Turing test". CNET. Retrieved May 10, 2018.
[4] Leviathan, Yaniv; Matias, Yossi (May 8, 2018). "Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone". Google AI Blog. Retrieved May 10, 2018.
[5] Carey, Bridget (May 9, 2018). "Human or bot? Google Duplex scares me". CNET. Retrieved May 10, 2018.
[6] Bergen, Mark (May 10, 2018). "Google Grapples With 'Horrifying' Reaction to Uncanny AI Tech". Bloomberg. Retrieved December 20, 2018.
[7] Welch, Chris (May 8, 2018). "Google just gave a stunning demo of Assistant making an actual phone call". The Verge. Retrieved May 20, 2018.
[8] HARDWICK, Tim. Siri's Multiple Language Support Still Its Biggest Strength Over Other Virtual Assistants. Macrumors.
[9] Oficiální stránka společnosti Apple ohledně Siri, https://www.apple.com/siri/
[10] DOČEKAL, Daniel. Jak do Androidu dostat Cortanu od Microsoftu? A co to umožní? [online]. 365tipu.wordpress.com, 1. 11. 2016
[11] SABRI, Sam. What do you think of Xiao Na?. Windows Central [online]
[12] ČÍŽEK, Jakub. Cortana míří do Francie a dalších zemí. Exotické jazyky mají smůlu. Živě.cz
[13] POLESNÝ, David. Cortana míří do byznysu, bude podporovat Microsoft Dynamics CRM 2015. Živě.cz
[14] "Alexa Voice Service Overview (v20160207) | Alexa Voice Service". developer.amazon.com.
[15] Turcan, Marie. "Test d'Amazon Echo : que vaut l'enceinte connectée d'Amazon en version française ?"
[16] "Angrez turns Desi: Amazon expands Alexa voice service to include Hindi". Business Insider.
[17] Sawers, Paul (15 November 2017). "Amazon brings Echo, Alexa, and Prime Music to Canada". VentureBeat. Retrieved 15 November 2017.
[18] Charron, François. "L'assistant vocal Alexa d'Amazon enfin disponible en québécois" (in French).
[19] Kinsella, Bret (2019-11-15). "Amazon Alexa Headcount Surpasses 10,000 Employees – Here is the Growth Rate". Voicebot.ai.
[20] Al-Heeti, Abrar (4 January 2019). "Amazon has sold more than 100 million Alexa devices". CNET. CBS Interactive. Retrieved 5 January 2019.
[21] Oficiální web Antelli, https://antelli.io
[22] Bc. Michal Blažek – „Implementace hlasového asistenta pro nevidomé“, 2017
[23] BAUM, L. E.; PETRIE, T. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. The Annals of Mathematical Statistics. 1966, s. 1554–1563
[24] Ed Grabianowski. How Speech Recognition Works, John Garofolo. How Stuff Works. 2016, http://electronics.howstuffworks.com/gadgets/high-tech-gadgets/speech-recognition.htm.
[25] Hoifung Poon a Pedro Domingos. Unsupervised semantic parsing. In: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1. 2009. 1–10.

[26] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov a Michael Collins. Globally Normalized Transition-Based Neural Networks. CoRR. 2016, abs/1603.06042
[27] ZELINKA, Ivan. Umělá inteligence: hrozba nebo naděje. 1. vyd. Praha: BEN - technická literatura, 2003, 142 s. ISBN 80-7300-068-7.
[28] Umělá inteligence - význam - IT Slovník [online]. Dostupné z http://it-slovnik.cz/pojem/umela-inteligence
[29] Turing test. In: Wikipedia: the free encyclopedia [online]. Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/turing_test
[30] H. A. Maarif, R. Akmeliawati, Z. Z. Htike a T. S. Gunawan. Complexity Algorithm Analysis for Edit Distance. In: 2014 International Conference on Computer and Communication Engineering. 2014. 135-137.
[31] ”Stavros Konstantinidis”. ”Computing the edit distance of a regular language”. ”Information and Computation”. ”2007”, ”205” (”9”), ”1307 - 1316”. DOI ”http://dx.doi.org/10.1016/j.ic.2007.06.001”.
[32] ”Giovanni Pighizzini”. ”How Hard Is Computing the Edit Distance?”. ”Information and Computation”. ”2001”, ”165” (”1”), ”1 - 13”. DOI ”http://dx.doi.org/10.1006/inco.2000.2914”.
[33] MEHRYAR MOHRI. EDIT-DISTANCE OF WEIGHTED AUTOMATA: GENERAL DEFINITIONS AND ALGORITHMS. International Journal of Foundations of Computer Science. 2003, 14 (06), 957-982. DOI 10.1142/S0129054103002114.
[34] Sandeep Hosangadi. Distance measures for sequences. arXiv preprint ar-Xiv:1208.5713. 2012
[35] L. Yujian a L. Bo. A Normalized Levenshtein Distance Metric. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2007, 29 (6), 1091-1095. DOI 10.1109/TPAMI.2007.1078.
[36] Alexandr Andoni a Krzysztof Onak. Approximating Edit Distance in Near-Linear Time. CoRR. 2011, abs/1109.5635
[37] Implementace hlasového asistenta pro nevidomé, Bc. Michal Blažek, ČVUT, 2017
[38] Allen, Jonathan; Hunnicutt, M. Sharon; Klatt, Dennis (1987). From Text to Speech: The MITalk system. Cambridge University Press. ISBN 978-0-521-30641-6.
[39] Alan W. Black, Perfect synthesis for all of the people all of the time. IEEE TTS Workshop 2002.
[40] John Kominek and Alan W. Black. (2003). CMU ARCTIC databases for speech synthesis. CMU-LTI-03-177. Language Technologies Institute, School of Computer Science, Carnegie Mellon University.
[41] Julia Zhang. Language Generation and Speech Synthesis in Dialogues for Language Learning, masters thesis
[42] William Yang Wang and Kallirroi Georgila. (2011). Automatic Detection of Unnatural Word-Level Segments in Unit-Selection Speech Synthesis, IEEE ASRU 2011.
[43] "Pitch-Synchronous Overlap and Add (PSOLA) Synthesis". Archived from the original on February 22, 2007. Retrieved 2008-05-28.
[44] T. Dutoit, V. Pagel, N. Pierret, F. Bataille, O. van der Vrecken. The MBROLA Project: Towards a set of high quality speech synthesizers of use for non commercial purposes. ICSLP Proceedings, 1996.
[45] Muralishankar, R; Ramakrishnan, A.G.; Prathibha, P (2004). "Modification of Pitch using DCT in the Source Domain". Speech Communication. 42 (2): 143–154. doi:10.1016/j.specom.2003.05.001.

[46] "Education: Marvel of The Bronx". Time. 1974-04-01. ISSN 0040-781X. Retrieved 2019-05-28.
[47] "1960 - Rudy the Robot - Michael Freeman (American)". cyberneticzoo.com. 2010-09-13. Retrieved 2019-05-23
[48] LLC, New York Media (1979-07-30). New York Magazine. New York Media, LLC.
[49] The Futurist. World Future Society. 1978. pp. 359, 360, 361.
[50] L.F. Lamel, J.L. Gauvain, B. Prouts, C. Bouhier, R. Boesch. Generation and Synthesis of Broadcast Messages, Proceedings ESCA-NATO
[51] Dartmouth College: Music and Computers Archived 2011-06-08 at the Wayback Machine, 1993.
[52] Examples include Astro Blaster, Space Fury, and Star Trek: Strategic Operations Simulator
[53] Examples include Star Wars, Firefox, Return of the Jedi, Road Runner, The Empire Strikes Back, Indiana Jones and the Temple of Doom, 720°, Gauntlet, Gauntlet II, A.P.B., Paperboy, RoadBlasters, Vindicators Part II, Escape from the Planet of the Robot Monsters.
[54] John Holmes and Wendy Holmes (2001). Speech Synthesis and Recognition (2nd ed.). CRC. ISBN 978-0-7484-0856-6.

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2020/2021

Studijní program: Aplikovaná informatika
Forma studia: Kombinovaná
Obor/kombinace: Aplikovaná informatika (ai3-k)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení: Dominik Palla
Osobní číslo: I1700298
Adresa: Ostrčilova 352, Trutnov – Dolní Předměstí, 54101 Trutnov 1, Česká republika

Téma práce: Česká hlasová asistentka Klára
Téma práce anglicky: Czech voice assistant Clara

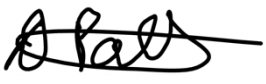
Vedoucí práce: prof. RNDr. PhDr. Antonín Slabý, CSc.

Zásady pro vypracování:

Existující realizace hlasových asistentů
Rozpoznávání řeči a převod na text
Syntéza řeči (TTS)
Výběr klienta pro účely práce (iOS)
Podstata, architektura, přístupy k soft řešení tohoto typu aplikací.
Implementace a testování
Závěr, zhodnocení, shrnutí

Seznam doporučené literatury:

1. Hoy, Matthew B. (2018). "Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants". *Medical Reference Services Quarterly*. **37** (1): 81?88. doi:10.1080/02763869.2018.1404391. PMID 29327988.
2. Klüwer, Tina. "From chatbots to dialog systems." Conversational agents and natural language interaction: Techniques and Effective Practices. IGI Global, 2011. 1-22.
3. Daniel B. Kline (2017-01-30). "Alexa, How Big Is Amazon's Echo?". *The Motley Fool*.
4. Artificial Intelligence through Prolog by Neil C. Rowe Prentice-Hall, 1988, ISBN 0-13-048679-5

Podpis studenta:  Datum: 26. 4. 2020

Podpis vedoucího práce:  Datum: 26. 4. 2020