
**UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
TÎRGU-MUREȘ
SPECIALIZAREA CALCULATOARE**

**PERSONAL ASSISTANT –
DESKTOP APP**

PROIECT DE DIPLOMĂ

**Coordonator științific:
Ș.l. dr. ing. Szántó Zoltán**

**Absolvent:
Magyarosi Roland-Mihály**

2021

UNIVERSITATEA "SAPIENTIA" din CLUJ-NAPOCA
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș
Specializarea: Calculatoare

Viza facultății:

LUCRARE DE DIPLOMĂ

Coordonator științific:
Ș.l. dr. ing. Szántó Zoltán

Candidat: Magyarosi Roland-Mihály
Anul absolvirii: 2021

a) Tema lucrării de licență:

Personal Assistant – Aplicatie Desktop

b) Problemele principale tratate:

- Studierea modurilor de pornire a aplicațiilor desktop
- Pregătirea datelor de tip audio
- Dezvoltarea unui algoritm care recunoaște utilizatorul după înregistrările video
- Dezvoltarea unui modul care poate executa următoarele comenzi: open firefox, search google, text, open notepad, search *.pdf, check mail, play video
- Dezvoltarea interfeței grafice prin care serviciile aplicației pot fi accesate

c) Desene obligatorii:

- Schema bloc al aplicației
- Diagrame UML privind software-ul realizat

d) Softuri obligatorii:

- Dezvoltarea unui modul care identifică comenzile vocale
- Dezvoltarea unei aplicații, care servește ca interfață cu utilizatorul

e) Bibliografia recomandată:

1. Yadgar, Osher, et al. "Generic virtual personal assistant platform." U.S. Patent No. 9,082,402. 14 Jul. 2015.
2. Natural-language voice-activated personal assistant
<https://patents.google.com/patent/US7216080B2/en>
3. Kalns, Edgar T., et al. "Rapid development of virtual personal assistant applications." U.S. Patent No. 9,081,411. 14 Jul. 2015.
4. Hauswald, Johann, et al. "Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers." *Proceedings of the Twentieth International Conference on nál hozzáfűztem még Architectural Support for Programming Languages and Operating Systems*. 2015.
5. Yadgar, Osher, et al. "Generic virtual personal assistant platform." U.S. Patent No. 10,163,440. 25 Dec. 2018.
6. Easwara Moorthy, Aarthi, and Kim-Phuong L. Vu. "Privacy concerns for use of voice activated personal assistant in the public space." *International Journal of Human-Computer Interaction* 31.4 (2015): 307-335.
7. Chen, Hongshen, et al. "A survey on dialogue systems: Recent advances and new frontiers." *Acm Sigkdd Explorations Newsletter* 19.2 (2017): 25-35.
8. Klopfenstein, Lorenz Cuno, et al. "The rise of bots: A survey of conversational interfaces, patterns, and paradigms." *Proceedings of the 2017 conference on designing interactive systems*. 2017.

f) Termene obligatorii de consultații: săptămânal

g) Locul și durata practicii: Universitatea Sapiientia,
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș

Primit tema la data de: 12.05.2020

Termen de predare: 28.06.2021

Semnătura coordonatorului

Semnătura candidatului

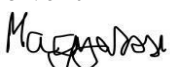
Declarație

Subsemnatul Magyarosi Roland-Mihály, absolvent al/a specializării Calculatoare, promoția 2021 cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea: Târgu Mureș

Data: 23.06.2021

Absolvent

Semnătura: 

Personal Assistant – DesktopApp

Extras

În zilele noastre, pe lângă dezvoltarea aplicațiilor web și telefonice, se acordă puțină atenție modernizării și regândirii aplicațiilor desktop, care joacă un rol foarte important în rândul utilizatorilor și al celor care lucrează în IT. Datorită dezvoltării rapide a telefoanelor de astăzi, practic totul poate fi rezolvat cu un dispozitiv inteligent în mâini și sângerând în buzunare, dar nu a reușit să elimine utilizarea computerelor și, odată cu acesta, aplicațiile desktop și probabil că va fi așa pentru ceva timp să vină.

Scopul disertației a fost de a implementa un sistem care rulează pe un computer desktop sau laptop care se potrivește perfect rolului unui asistent personal virtual în viața noastră. Așa-numitul software de asistent personal inteligent (IPA) ajută utilizatorii să efectueze sarcini simple, dar obișnuite, cum ar fi navigarea pe Internet, trimiterea rapidă a e-mailurilor și deschiderea rapidă a aplicațiilor pe o mașină, nu prin introducerea codurilor și a semnalelor mașinii, ci pur și simplu în limba. Se crede că această naturalețe a adus și va aduce până în prezent succesul științei care a devenit o industrie.

Asistenții virtuali sunt o binecuvântare pentru umanitate în secolul 21, dar istoria lor nu a început aici, surprinzător datând din anii 1920. Există atât de multe tipuri de asistenți personali, dar ideea este aceeași: pentru a servi utilizatorului, în cazul nostru utilizatorul poate fi o persoană care lucrează în fața unui computer al cărui software numit Sapia Sarah încearcă să ușureze lucrurile executând lucruri simple, dar comenzi utilizate frecvent. Această tehnologie este disponibilă atât pe smartphone-uri, cât și pe computere, dar apare acum și în electrocasnice și mașini.

Disertația mea va acoperi aplicațiile desktop și proiectarea lor, recunoașterea vorbirii, executarea comenzilor bazate pe voce și asistenții virtuali, frecvența, structura, utilizarea și dezvoltarea acestora.

Cuvinte de cheie : asistent virtual, aplicație desktop, utilizator, tehnologie.

**SAPIENTIA ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR
SZÁMÍTÁSTECHNIKA SZAK**

**VIRTUÁLIS ASSZISZTENS –
ASZTALI ALKALMAZÁS**

DIPLOMADOLGOZAT

Témavezető: S.I. dr. ing. Szántó Zoltán

Végzős hallgató: Magyarosi Roland-Mihály

2021

Kivonat

Napjainkban a web és telefonalkalmazások fejlesztése mellett kevés figyelem hárul az asztali applikációk korszerűsítésére, újragondolására, amely igencsak fontos szerepet lát el a felhasználók, illetve az IT-ban dolgozók körében. A telefonok rohamos fejlődésének hála manapság már gyakorlatilag minden megoldható a kezünkben levő és a zsebünkben lévő okos eszközzel, de a számítógépek és ezzel együtt a Desktop alkalmazások használatát nem tudta eltörölni és valószínűleg ez egy ideig még így lesz.

A dolgozat egy olyan asztali számítógépen, vagy laptopon futó rendszer megvalósítását tűzte ki célul, amely tökéletesen megfelel a virtuális személyi asszisztens szerepkörnek az életünkben. Az úgynevezett intelligent personal assistant (IPA) szoftverek segítik a felhasználókat egyszerű, ám gyakori feladatok elvégzésében, ilyen például az interneten való böngészés, gyors e-mail küldés, valamint a gépen való alkalmazások gyors megnyitása, mindezt nem gépi kódok és jelek bevitelével, hanem egyszerűen, emberi nyelven. Vélhetően ez a természetesség hozta és hozza mai napig az iparággá nőtt tudomány sikerét.

A virtuális asszisztensek a 21. század áldása az emberiség számára, ám a történelmük nem itt kezdődött, meglepő módon egészen az 1920-as évekig nyúlik vissza. Ahány személyi asszisztens annyi féle, ám a lényegük ugyanaz: a felhasználó kiszolgálása, esetünkben a felhasználó egy számítógép előtt dolgozó ember lehet, akinek a Sapientia Sarah névre keresztelt szoftver igyekszik megkönnyíteni a dolgát egyszerű ám, bár gyakran használt parancsok végrehajtásával. Ez a technológia elérhető okostelefonokon és számítógépeken egyaránt, de mára már a háztartási gépekben és az autókban is feltűnik.

A dolgozatunkban az asztali alkalmazásokról és azok tervezéséről, a beszédfelismerésről, a hangalapú parancsok végrehajtásról, illetve a virtuális asszisztensekről szóló tanulmányozást olvashatnak, azok gyakoriságáról, felépítéséről, hasznáról és fejlesztéséről.

Kulcsszavak: virtuális asszisztens, asztali alkalmazás, felhasználó, technológia

Abstract

Nowadays less and less attention is given to development and modernizing of Desktop applications over web and mobile. Although Desktop applications have an important role in the lives of users and IT workers. With the quick evolution of mobile phones practically everything is possible using the smartphone in our pocket, but it couldn't and probably will not be able to surpass computers for a while.

The dissertation focuses on a system designed for personal computers or laptops that perfectly meets the role of a virtual assistant in our life. The IPA (intelligent personal assistant) softwares help the uses perform easy, but frequent tasks like browsing the internet, sending e-mails and swiftly opening applications while the input is not codes or signals, but simply a human voice. Presumably this is the reason for the success of this science that has grown into an industry.

Virtual assistants are the blessing of the 21st century, but their history dates back to 1920. Every assistant is different, but their purpose is the same: serving the user. In our case a user is person in front of a computer whose job Sapiientia Sarah tries to make easier, with performing simple, but frequent commands. This technology is accessible on both smartphones, computers and recently in household appliances and cars as well.

In my dissertation I studied desktop applications and development of these, voice recognition, executing voice commands and the structure, use and development of virtual assistants. Keywords: virtual assistant, desktop application, user, technology

Keywords: virtual assistant, desktop application, user, technology

Tartalomjegyzék

Ábrák, táblázatok jegyzéke	5
Kódrészletek jegyzéke	6
1. Bevezető.....	7
1.1. Bevezető.....	7
1.2. A dolgozat témája.....	7
1.3. Célkitűzések.....	9
2. Bibliográfiai tanulmány	11
2.1. Siri	12
2.2. Mycroft.....	12
2.3. Cortana.....	14
2.4. Megszorítások.....	16
2.5. Következtetés.....	17
3. Követelmény specifikáció	18
3.1. Felhasználói követelmények.....	19
3.2. Rendszerekövetelmények	26
3.2.1. Funkcionális követelmények	26
3.2.2. Nem-funkcionális követelmények.....	28
4. A rendszer architektúrája	31
4.1. A felhasználó	33
4.2. A felhasználói felület	33
4.2.1. A forrásanyagok	38
4.2.2. Írásos kimenet	38
4.3. A hangbemenet és beszédfeldolgozás	49
4.3.1. Áttekintés	49
4.3.2. A „SpeechRecognition” könyvtár	50
4.3.3. Az asszisztens válasza, Sarah hangja	53
4.4. Brain modulok	55
4.4.1. Idővel kapcsolatos	55
4.4.2. Internettel kapcsolatos.....	56
4.4.3. Rendszerrel kapcsolatos	60
4.4.4. Időjárással kapcsolatos.....	63
4.5. Kriptográfia.....	64
4.5.1. Áttekintés	64
4.5.2. A Fernet módszer	65
5. Üzembe helyezés.....	67
5.1. Felmerülő problémák és megoldásaik.....	67
6. Következtetések.....	68
6.1. Megvalósítások.....	68
6.2. További fejlesztési irányok.....	69
7. Irodalomjegyzék.....	70

Ábrák, táblázatok jegyzéke

1. Ábra A legfontosabb hang vezérelt platform	8
2. Ábra: Telefonos személyi asszisztensek népszerűsége	8
3. Ábra: A legjobb virtuális asszisztens [1]	9
4. Ábra: Spektrogram a „virtuális asszisztens” inputra.....	13
5. Ábra: A legjobb keresőmotorok.....	14
6. Ábra: Cortana leggyakoribb végrehajtások.....	16
7. Ábra: A háromszöges probléma.....	18
8. Ábra: Az alkalmazás Használat-Eset diagramja	20
9. Ábra: A rendszer architektúrája	31
10. Ábra: Sarah logója	34
11. Ábra: A komponens diagram	40
12. Ábra: A Main Screen	41
13. Ábra: A Volume Selector	42
14. Ábra: Az About Us ablak	43
15. Ábra: A Privacy Policy ablak.....	44
16. Ábra: A Send E-mail ablak	45
17. Ábra: Az adatok módosítása	47
18. Ábra: Az időjárás widget aktiválás előtt	48
19. Ábra: Az időjárás widget aktiválás után.....	49

Kódrészletek jegyzéke

1. Kódrészlet: Egy munkásszál	36
2. Kódrészlet: Egy GUI indítása	37
3. Kódrészlet: A szövegdobozba írás.....	39
4. Kódrészlet:A Main Screen Start gombja.....	42
5. Kódrészlet: A Volume Selector gombjai.....	43
6. Kódrészlet: Az e-mail küldés	46
7. Kódrészlet: A hangbemenet elcsípése	51
8. Kódrészlet: A hangbemenet értelmezése.....	52
9. Kódrészlet: A feldolgozó szál elindító QThread osztály	53
10. Kódrészlet: A beszédért felelős metódus	54
11. Kódrészlet: A TimeC osztály teljes tartalma	55
12. Kódrészlet: Az InternetC osztály első metódusa	56
13. Kódrészlet: A nagy vállalatok értékének megkeresése	57
14. Kódrészlet: Keresés az interneten	58
15. Kódrészlet: Navigálás a Google Maps segítségével.....	59
16. Kódrészlet: Kilépés a rendszerből	60
17. Kódrészlet: A köszöntés	60
18. Kódrészlet: A rendszer lenémítése	61
19. Kódrészlet: Az alkalmazások megnyitása	61
20. Kódrészlet: Időjárás lekérés a felhőből	64
21. Kódrészlet: A Crypto komponens osztálya.....	66
22. Kódrészlet: A nyers anyag kinyerése a config fájlból.....	67

1. Bevezető

1.1. Bevezető

Napjainkban a web és a telefonalkalmazások fejlesztése mellett kevés reflektorfény hárul az igen fontosnak mondható asztali applikációk fejlesztésére, amely nélkülözhetetlen szerepet lát el az átlagfelhasználók, illetve az IT-ban dolgozók számára egyaránt. Egy felmérés szerint¹ világviszonylatban évről évre leginkább a Desktop fejlesztéssel foglalkozó nyelvek stagnálnak, ezzel ellentétben a web, illetve a telefon applikációk fejlesztése szárnyal. Mindez betudható annak is, hogy a telefonok fejlődésével és a kijelzők méretének és minőségének javulásával az interneten való böngészést is leginkább a telefonjukon oldják meg a felhasználók, ugyanis kényelmes és a zsebből előrándítható.

Manapság elavult a régen hatalmas innovációnak számító billentyűzet, egér kombináció, a mai rohanó világban mindent a legegyszerűbb módon szeretnénk végrehajtani, az sincs kizárva, hogy egyszerre több dolgot szeretnénk csinálni. Mára az már teljesen megszokottnak számít, hogy az okos eszközöknél beköszönve arra utasítod azt vezetés közben, hogy telefonhívást indítson, vagy üzenetet küldjön, az okos otthonokat előre lehet fűteni, világítani, sötétíteni stb. Kezdetben a tableteknek és okostelefonoknak hála az érintőképernyő váltotta a gombokat, mára az is teljesen megszokott már, az új trend a hang alapú végrehajtás.

A dolgozatomban egy olyan rendszert mutatok be, ami a Laptop, illetve asztali számítógép beépített, vagy hozzacsatolt kameráját és mikrofonját felhasználva adatot gyűjt és parancsokat hajt végre. A kamera felismeri a gép előtt ülő felhasználót, ha az a rendszer előre megadott tulajdonosa, akkor hang alapú végrehajtást tesz lehetővé, hasonlóan a nagynevű ilyen jellegű alkalmazásokhoz, amelyek egy része bemutatásra kerül a dolgozat későbbi részében.

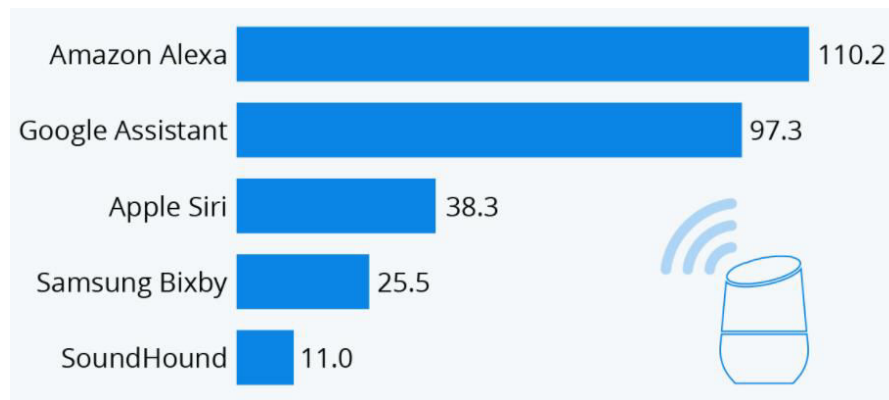
1.2. A dolgozat témája

A dolgozat témája egy személyi asszisztens jellegű alkalmazás. Az úgynevezett intelligent personal assistant (IPA) szoftverek segítik az átlag felhasználókat egyszerű ám gyakori feladatok elvégzésében, mindezt emberi nyelven, gépi kódok és kódolt jelek bemenete nélkül, ez a természetesség és emberi közelség hozza leginkább az iparággá nőtt tudomány sikerét.

A személyi asszisztenseknek sok különféle formáját dobták piacra, ám a lényegük ugyan az: a felhasználó kiszolgálása. A legnépszerűbbek többek között a Google által fejlesztett Google

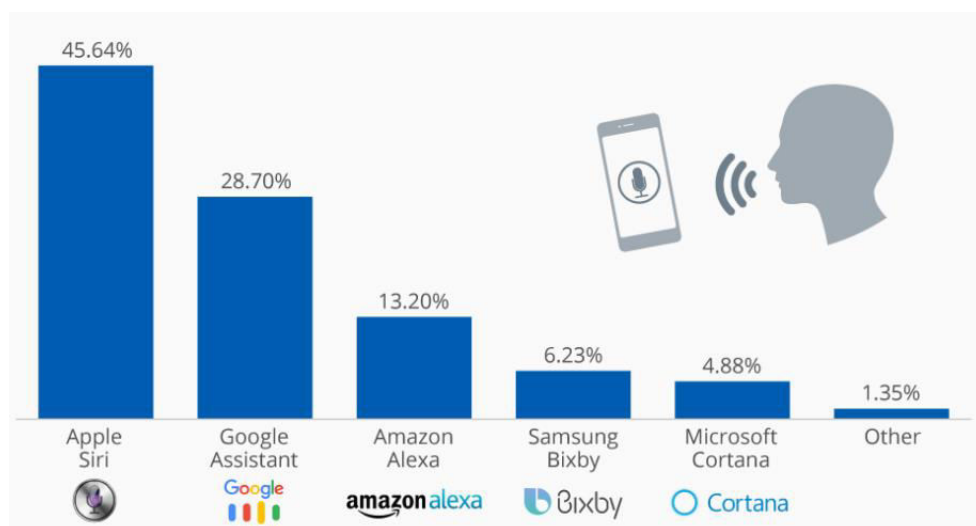
¹ Development trends <https://www.daxx.com/blog/development-trends/number-software-developers-world>

Now, az Apple Sirije, a Microsoft féle Cortana és az Amazon Alexa. Egy felmérés szerint² a legfontosabb hang vezérelt platform az nem más, mint az Amazon Alexa, ezen statisztika szemléltetésre került az 1-es ábrán.



1. Ábra A legfontosabb hang vezérelt platform ²

Ugyanezen forrás adatai szerint a leghasználtabb mind közül az Apple fejlesztésű Siri, amely akár üzenetküldésre, hívásra, ébresztő beállításra is alkalmas, a cég közlése szerint a 1.4 milliárd Apple felhasználóból több, mint az egy harmada, azaz nagyjából 500 millió embernek könnyíti meg napi szinten a Siri a dolgát hangvezérléssel. A Siri volt egyébként az első okostelefonba beépített intelligens asszisztens, 2011 októberéből. A 2-es ábrán látható a Siri abszolút fölénye, ami a telefon alkalmazásokban használt személyes asszisztensek népszerűségét jelzi.



2. Ábra: Telefonos személyi asszisztensek népszerűsége²

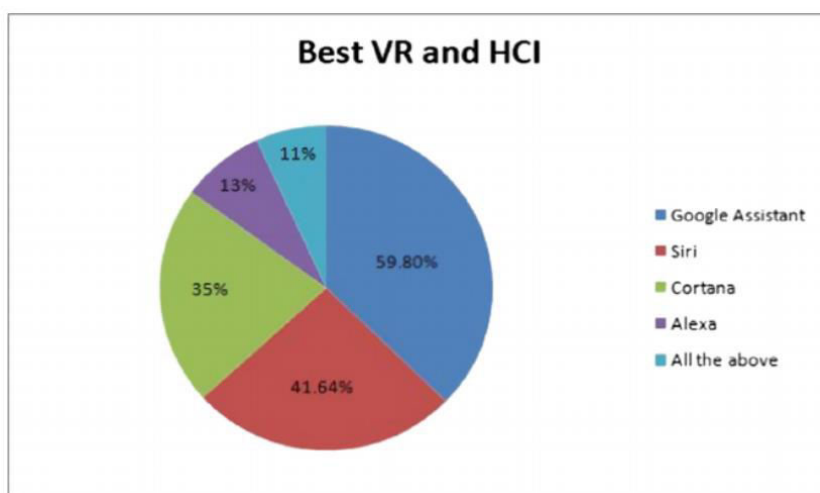
Egyes felmérések szerint³ a felnőttek 40%-a vallja azt, hogy napi szinten használ valami féle hang alapú keresést, a hagyományos gépelés helyett, ugyanez a hírforrás állítása szerint Cortanának 133 millió felhasználója van havonta. Az emberek 72%-a, amely rendelkezik

² Voice platform ranking: <https://www.statista.com/chart/22314/voice-platform-ranking/>

³ IPA usage stats: <https://edit.co.uk/blog/google-voice-search-stats-growth-trends/>

hangalapú asszisztenssel napi rendszerességgel használja azt, sőt mi több azt vallják, hogy a napi rutinjukhoz is hozzátartozik mindez.

Egy kutatás [1] szerint, amely a legjobb virtuális asszisztent kereste a Google Assistantról, Siritől, Cortanáról, illetve Amazon Alexáról kérdezte az alanyokat, 100 felhasználót, amelyből 55 nő, illetve 30 folyamatos felhasználója valamelyik virtuális asszisztensnek, 70 pedig csak gyakori. Ezen felhasználók egy kérdést kaptak tömördek témából, amelyet fel kellett tenniük mindegyik asszisztensnek külön-külön, ezt követően kiértékelni a válaszokat és választani 1-et. Ahogyan az a 3. ábrából is kivehető, a Google Assistant nyerte le leginkább az alanyok tetszését.



3. Ábra: A legjobb virtuális asszisztens [1]

Az alkalmazás kellően biztonságos lenne, amennyiben létrejött volna a társprojekt, ahol az arc alapú felismerés a tesztfázisból kilépve azonosítaná a felhasználóját. Az ún. Ultra Light modell-t teszteltük korábban, amely kellően gyors és pontos ahhoz, hogy a másodperc töredéke alatt megismerje a felhasználóját. Az általunk tesztelt három módszer (Ultra Light, Mtcnn ill. Hog) közül az első alkalmazására esett a választásunk, ugyanis megfelelően pontos és gyors több főből álló csoportképek arc detekciójára is. Méréseink szerint 100%-os pontossággal 0.056 szekundum alatt ismer fel az algoritmus egy 12 fős csoportkép arcait.

1.3. Célkitűzések

A dolgozat célja egy személyi asszisztens jellegű szoftver elkészítése, amely képes a felhasználó kimondott parancsait végrehajtani, ezzel is rövidítve és hatékonyabbá téve a számítógép előtt ülő személy munkáját. Nem titok, hogy az alkalmazásunk legfőképpen a gép

előtt dolgozó emberek kiszolgálását tűzte ki célul, hogy azok minél egyszerűbben érhék el az egyszerű, ám a mindennapokban használt számítógépes funkciókat.

A különféle funkcionalitások olyan parancsokat hajtanak majd végre, hogy felismert parancsot követően megnyitja a böngészőt, sőt egyéb asztali applikációkat is, az interneten való keresés, a pontos idő/dátum lekérése, különféle alkalmazások megnyitása, gyors e-mail küldés és ezekhez hasonló funkciók. A projekt két összefüggő részből áll: egy felhasználó azonosítása kép alapján, másrészt hang alapú parancsok felismerése és a megfelelő konfigurációs alkalmazás elkészítése. A dolgozatban a második részével foglalkozunk, melyben tulajdonképpen a projekthez egy felhasználói felületet fejlesztünk Desktop applikáció formájában, valamint a hangfelismeréssel is foglalkozunk és a hang alapú végrehajtással.

Napjainkban a web és telefonalkalmazások fejlesztése mellett viszonylag kevés reflektorfény hárul az asztali applikációk fejlesztésére, korszerűsítésére, újragondolására, amelyek rendkívül fontos szerepet látnak el a felhasználók, illetve az IT-ban dolgozók körében egyaránt. Annak ellenére, hogy a telefonos alkalmazások rohamos fejlődésével mára minden elérhető a zsebünkben rejtőző okoseszköz használatával, ám vélhetően az egy ideig nem törli el semmi az asztali programok fontosságát, főleg annak tudatában, hogy ezen mobileszközök programozása is Desktopon történik a legkényelmesebben.

Céljaink közé tartozik egy olyan modern design-nal ellátott alkalmazás fejlesztése, amely eladható, felhasználóbarát és átlátható, könnyen kezelhető mindazok mellett, hogy minden kellő funkcióval rendelkezik ahhoz, hogy kiszolgálja az adott felhasználói tábor, illetve a mögötte rejtőző backendet.

Az úgynevezett backend az elképzelések szerint hangértelmezésből, illetve végrehajtásból fog állni. Egy olyan algoritmust kell kidolgozni, amely egyszerű és kellően optimális ahhoz, hogy ezen feladatokat képes legyen ellátni.

Az alkalmazáson keresztül a felhasználó a szokásos beállításokon (profil, mikrofon, hangerő) kívül kezelheti a már meglévő parancsokat. További céljaink közé tartozik a felhasználó idejének spórolása mellett a megbízhatóság és a védelem, az arcfelismerésnek hála az alkalmazás azonnal észleli, ha nem a tulajdonos ül a számítógép, vagy a laptop előtt.

2. Bibliográfiai tanulmány

A hangalapú felismerés tudományág történelme egészen messze, az 1920-as évekig nyúlik vissza, mikor is az Elmwood Button Co. kifejlesztett egy hangdetekción alapuló játékot, amely a „Radio Rex”⁴ nevet kapta. A viszonylag egyszerű játék arról szólt, hogy egy bulldog a neve hallatán kiszalad a fa házából. A kutyaházból való kilökését egy rugó tette lehetővé, amelyet egy 500 Hz-es akusztikus energia felszabadított. Mivel a „Rex” szócska nagyjából 500 Hz-en mozog a tartományban az „e” magánhangzónak köszönhetően úgy tűnt, hogy a játékszer akkor bújik elő az odújából, amikor hívják. A probléma a megközelítéssel annyi csupán, hogy a mély férfi hang „e” magánhangzója mozog körül belül az 500 Hz-es skálán, a női és a gyerek hang „e” magánhangzója megközelítőleg 740 Hz-en mozog, így annak érdekében, hogy nekik is kijöjjön a házból a kutya nagyon hangosan és mélyen kell szólítaniuk őt, továbbá az is egy lehetőség, hogy „Rix”-nek, illetve „Rux”-nak becézik, mivel az „i” meg az „u” magánhangzó csengése jóval alacsonyabban mozog az „e”-nél.

Hatalmas áttörésnek számít a szakmában az IBM Shoebox fejlesztésű hang-aktivált számológép, amely 1962-ben került a közszemlé⁵ elé. A rendszer már jelen volt az első asztali gép megjelenése előtt 20 évvel. Az innovációnak számító alkalmazás képes volt nullától kilencig felismerni a számokat, emellett azon utasításokat is végre hajtotta, mint a „plus” a „minus”, illetve a „total”. A Shoebox 16 különféle szó felismerésére volt alkalmas, ezen fejlesztés nagyban hasonlít a dolgozat témájához. A számológép hang bemeneten alapult, tehát egy mikrofonra volt szükség, amely a bemenetet jelentette, ezen bemenet elektromos impulzusokká alakult át, tehát jeleket kezelt. Egy mérőáramkör ezen impulzusokat különféle hangtípusok szerint osztályozta, ezek összessége alkotta meg a rendszert.

1966-ban először megteremtődött a kapcsolat az ember, és a gépek között, ugyanis akkor először tudott kommunikálni egymással emberi nyelven a két fél a Chatbot Elizának⁶ hála – hatalmas lépésként és egyenesen innovációként élték meg a megjelentését a tudósok és máig áttörésként említik. Eliza input feldolgozása kulcsszavak elemzésével kezdődik, tehát hierarchia volt felfedezhető a szavak között. Amikor a rendszer felismerte a kulcsszavakat, akkor rang szerint rendezte. Ezen rendezést az átalakítási szabály első módszere, a bontás előzte meg, amikor a mondatot szétbontva feldolgozta hierarchikusan a kulcsszavakat, ezt követően jött a szabály

⁴ Radio Rex: <https://ileriseviye.wordpress.com/2011/02/17/speech-recognition-in-1920s-radio-rex-the-first-speech-recognition-machine/>

⁵ IBM calculator: https://www.ibm.com/ibm/history/exhibits/specialprod1/specialprod1_7.html

⁶ Chatbot Eliza: <https://medium.com/nlp-chatbot-survey/computational-linguistics-754c16fc7355>

második fele: az összeszerelés. A rekonstruálás megannyi szabály szerint működött, hogy a bevitelre a felhasználó értelmes és különleges választ kapjon.

2.1. Siri

Napjaink egyik leghasználtabb és legjobban értékelt virtuális személyi asszisztense nem más, mint az Apple féle Siri. A szoftver sikerének titka leginkább annak tudható be, hogy nem csak a telefonon, vagy az asztali gépen elérhető, hanem az Apple összes termékében helyet kapott, az iPhoneban, az iPadban, az iPodban, a MacOS-ban, az AirPodsban, a HomePodban és az Apple TV-ben egyaránt. Siri 2011 október 12.-én debütált az iPhone 4S-ben, ezzel az első okostelefonba épített virtuális intelligens asszisztensé avanzsált. [1]

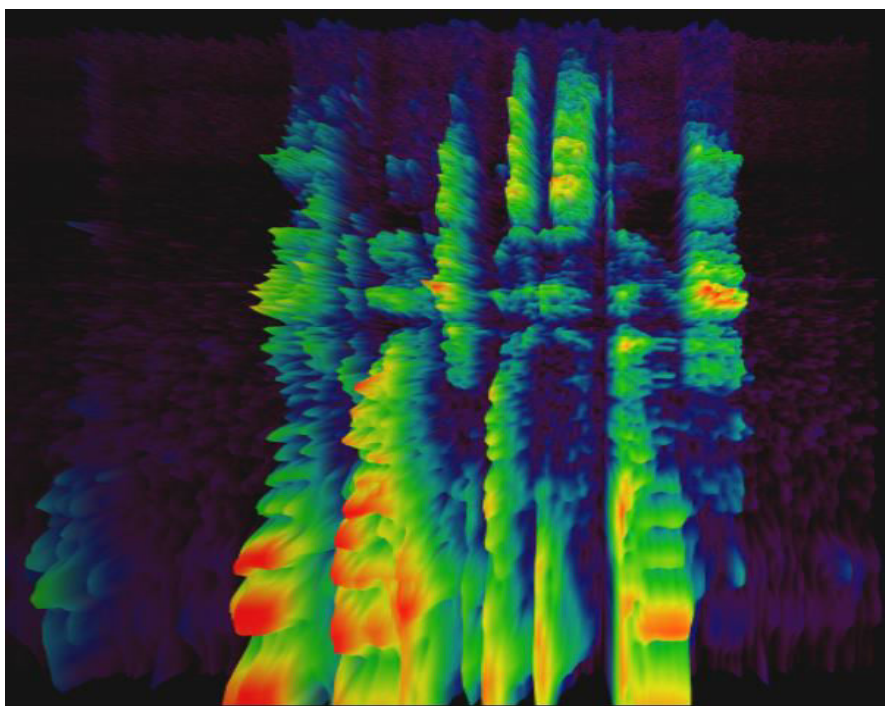
A manapság jelen levő adat-kezelő „háborúban” az Apple minden egyes felhasználóját biztosítja afelől, hogy a Siri csak a saját maga fejlesztésére használja fel a privát adatokat és a cég nem él vissza vele. A cég biztosít mindenkit afelől, hogy a program a legkevesebb adatot veszi igénybe fel a pontos eredmény eléréséhez, ha például egy sportesemény felől érdeklődünk, a Siri az általános tartózkodási helyünket használja fel a megfelelő eredmények biztosításához, de ha a legközelebbi élelmiszerboltot keressük, akkor konkrét hely adatokat használ. Egyéb példaként szolgál a Siri arra utasítása, hogy olvassa fel az olvasatlan üzeneteket. Ebben az esetben a szoftver végrehajtja az utasítást, ám az üzenet tartalmának a minimális része sem kerül fel a szerverekre, ugyanis erre nincs szüksége a végrehajtáshoz. [2]

Siri működésének⁷ alapelve kísértetiesen hasonló a fent említett, 1962-ben fejlesztett számológépéhez: a beépített mikrofonon hangot vár, majd azt alakítja át a gép által érthető jelekké, majd következik a végrehajtás. A fejlesztők az Eliza féle algoritmuson is vélhetően átrágták magukat a tervezési fázisban, ugyanis rengeteg hasonlóság fedezhető fel a két fejlesztés között: az input lekódolása után a szoftver lebontja a kódot, azonosítja az egyes mintákat, kifejezéseket és kulcsszavakat, amelyeket hierarchikusan kezel. Ezen adatok egy algoritmusba kerülnek, amely több ezer mondatkombinációt használ fel, hogy meghatározza mit jelent a bevitt kifejezés. Az imént említett algoritmus elég komplikált ahhoz, hogy kellően megkülönböztesse, vagy esetleg egy csoportba sorolja az idiómákat, illetve a homofonokat és elég titkos ahhoz, hogy senki ne tudja felhasználni.

⁷ Siri works: <https://www.jameco.com/Jameco/workshop/Howitworks/how-siri-works.html#:~:text=Upon%20receiving%20your%20request%2C%20Siri,patterns%2C%20phrases%2C%20and%20keywords.>

2.2.Mycroft

A Mycroft egy nyílt, privát hangkezeléssel működő asszisztens (utalás a szlogenre). A nyíltságát a forráskódja adja meg, amely mindenki számára elérhető, a személyessége a kérések végrehajtásában rejlik. Az esetünkben a Mycroft azért is érdekes, mert Pythonban íródott, így megegyezik a célnyelvünkkel. Az alkalmazás gépi tanuláson alapszik, a gép „látja” a hangot. A számítógépes hallás lényegében egy vizuális folyamat, spektrogram jeleníti meg a beszédet a gépnek, amely különféle frekvenciákat izolál és jelzi, hogy az egyes hanghullámok energiája mennyi ideig tartott. A spektrogram szemléltetésképpen megtalálható a 4-es ábrán, amely generálása egy online generátort⁸ használtam, a „virtuális asszisztens” inputra. [3]



4. Ábra: Spektrogram a „virtuális asszisztens” inputra⁸

A Mycroft számos nyílt forráskódú hardvert és szoftvert használ fel a mesterséges intelligencián alapuló platformjához. A készülék középpontjában egy nagyteljesítményű Raspberry Pi 2 található. Az 1 GB memóriával rendelkező, négymagos ARM CortexA7 processzoron alapuló „keménymag” elég erős ahhoz, hogy a platform motorja legyen. Az állapotot mutató kijelzőt a népszerű Arduino platform hajtja.

Miután a készülék csatlakozott az otthoni WiFi hálózathoz több különféle otthoni okos eszközökkel tud kapcsolatot létesíteni, „beszélgetni”. Számos olyan online szolgáltatást is támogat, mint a YouTube, Netflix, Pandora, Spotify és hasonlóak. A beépített hangszóróknak hála zene lejátszásra is kérheti a felhasználó, minden megkötés nélkül.

⁸ Spektrogram: <https://musiclab.chromeexperiments.com/Spectrogram/>

Ez a fejezet tartalmazza a hasonló alkalmazások, megoldások rövid ismertetőjét, amelyek már léteznek. Ugyan csak ide kerül azoknak a technológiáknak, elméleti ismereteknek a rövid bemutatása, ami szükséges a megvalósítás megértéséhez.

2.3.Cortana

A Microsoft saját fejlesztésű virtuális intelligens asszisztense a Cortana névre hallgat. Esetünkben azért érdekes, mert a szoftver legfőbb alkalmazása Windows alatt fut asztali gépen, vagy laptopon. A Windows telefonok „kihalásával” a Cortana fejlesztése leginkább desktop irányba haladt, ezért is gondolják manapság úgy a cég elöljárói, hogy az asszisztensük nincs egy szinten például Alexával. Az asszisztens nem kizárólag hang alapú végrehajtást tesz lehetővé, ugyanis Windows alapon a kellő billentyűkombinációk lenyomásával a készülékünkön lokális keresést végez, de a Bing segítségével a keresést az internetre is kiterjesztheti, amolyan kereső motorként. Érdekességgéppen a Google a piacon körül belül 92%-os részesedést tudhat magáénak e tekintetben, a Bing kevesebb, mint 3%-kal rendelkezik⁹. A legjobb tíz keresőmotor listáját az imént említett felmérés szerint a 5. ábra is szemlélteti.

THE TOP 10 SEARCH ENGINES	
SEARCH ENGINE	MARKET SHARE
1. GOOGLE	91.54%
2. BING	2.44%
3. YAHOO!	1.64%
4. BAIDU	1.08%
5. YANDEX	0.54%
6. DUCKDUCKGO	0.45%
7. SOGOU	0.44%
8. ECOSIA	0.14%
9. SHENMA	0.08%
10. NAVER	0.07%

5. Ábra: A legjobb keresőmotorok⁹

⁹ Search market statistics: <https://www.webfx.com/blog/seo/2019-search-market-share/>

A „Hey, Cortana” kombináció hallatán Radio Rexhez és Sirihez hasonlóan az asszisztens felébred és várja a felhasználó kéréseit, hogy végrehajthassa azt.

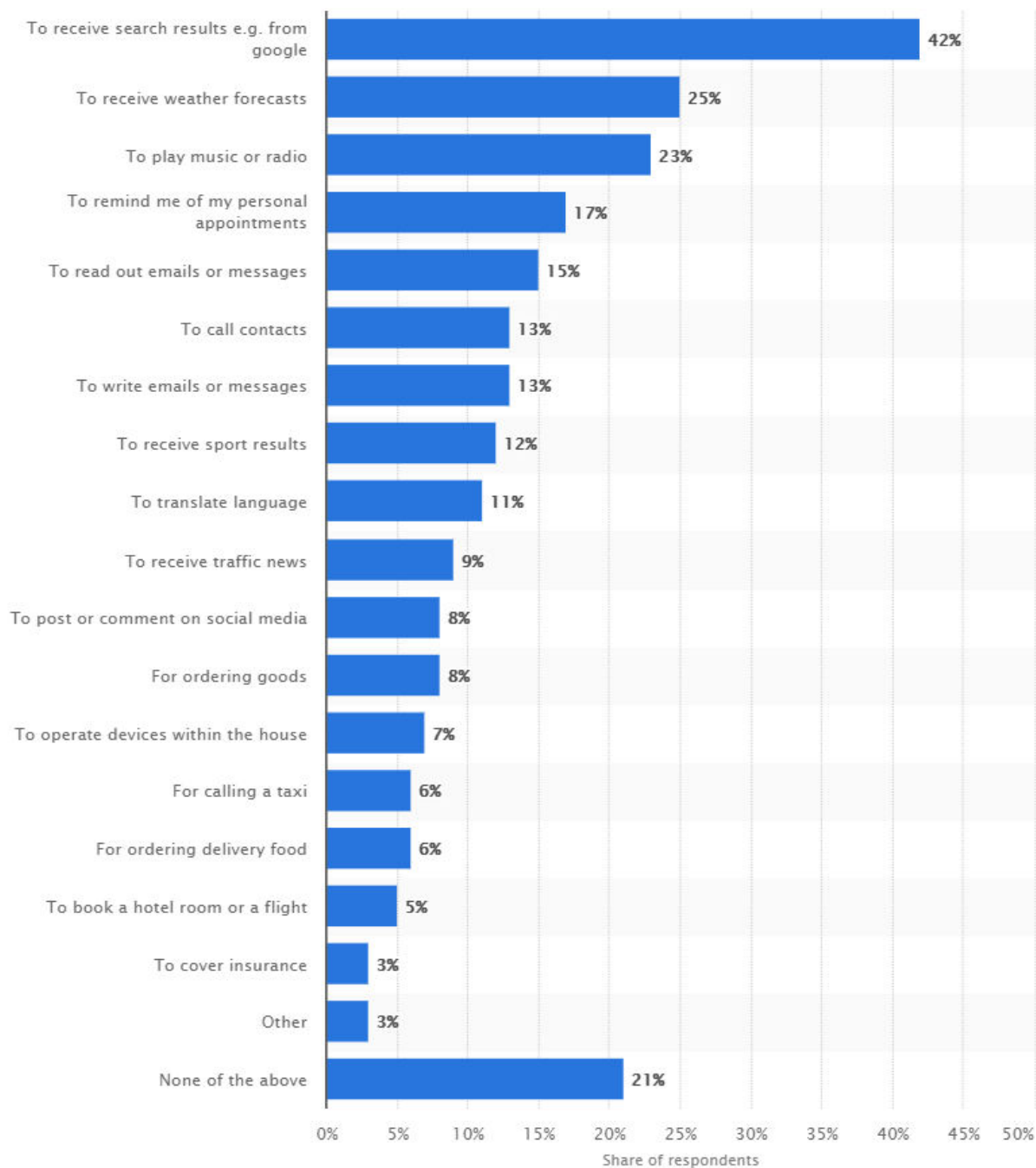
Cortana fájlokat és mappákat kereshet akár közvetlenül az asztalról, de a felhőhöz is van hozzáférése – ilyen például a OneDrive. Az alkalmazás is a saját adataid felhasználásával pontosítja az általa közölt információt – ilyen lehet a helymeghatározás az időjárás előrejelzéshez.

A Notebook a Cortana egyik érdekes integrációs szolgáltatása, amelyre kattintva megtaláljuk a személyre szabott érdeklődési körünket, amelyet az algoritmus az internetes kereséseink alapján javasol. Cortana ellenőrizheti a felhasználó e-mailjeit és naptárját, kéretlen értesítéseket küldhet. Nagyon szellemes „személyiséggel” rendelkezik, vicceket tud mesélni, úgy van programozva, hogy változékonyan válaszoljon a kérdésekre a monotonitás elkerülése végett.

Cortana bele van építve a Microsoft saját fejlesztésű böngészőjébe, a Microsoft Edgebe, ahol az eredményes böngészésben segít, továbbá spontán kérdésekre gyors válaszokat ad, emellett az Xboxban is fellelhető. [4]

A virtuális asszisztenseket a mindennapi, általános, rutinszerű problémák könnyen elvégzésére használják, de hogy mik is ezek a problémák? Egyes statisztikai adatok¹⁰ szerint a Microsoft Cortanáját a felhasználók leginkább Google-n belüli keresésre, időelőrejelzésre, zene lejátszásra használták, a teljes felmérés eredményét a 6-os ábrán megfigyelhetjük.

¹⁰ Cortana Usage: <https://www.statista.com/forecasts/1038106/usage-of-microsoft-s-cortana-s-functions-in-the-us>



6. Ábra: Cortana leggyakoribb végrehajtások⁹

2.4.Megszorítások

Ezen rendszerek egyik nagy korlátja a hatalmas mennyiségű információ, amelyet el kell tárolni valahol – esetünkben a felhőben tárolják az adatot a beépített könyvtárak. A profi, világvezető virtuális asszisztensek megannyi különféle kifejezést, szót kell eltárolnia a szervereiken, ezeknek nem annyi információt kell tartalmazniuk, mint a korábban kivesézett

Radio Rexnek, vagy esetleg a IBM Shoeboxnak, fel kell tudnia ismerni a különféle idiómákat, illetve a homofonokat, természetesen ezeket is kellően tárolva.

Az egyik nagy korlát az az optimális működés, illetve a gyorsaság. A mai rohanó világban az emberek mindent azonnal akarnak, nagyon gyorsan szeretnék tudni a választ a holnapi időjárásra, vagy a tegnapi meccs eredményét is azonnal tudni akarják. Az asszisztensek nagyon gyorsak kell legyenek, kevés idejük van feldolgozni az inputokat, emellé megfogalmazni a tökéletes és helyes, esetenként változatos választ.

Nagy port kavar manapság a Sirinél említett jog feldolgozás, illetve adat felhasználás. Mára már tabuvá vált a személyes adatok eladása, közlése, mindenki tudja, hogy jelen van, de senki nem hajlandó beszélni róla. Ezek ellenére nagy üzlet és heti, vagy havi rendszerességgel bírságnak cégeket a személyi jogok megsértése miatt. Egy személyes asszisztens telepítésével fel kell készülnünk arra, hogy az adott algoritmus az összes készülékünkön levő adathoz hozzáférhet, ha esetleg úgy van programozva. Ezen korlát első sorban az alkalmazás marketing értékét és a belé vetett bizalmat gátolja, nem sorolandó a funkcionális megszorítások közé a mikrofonnal, kamerával stb. ellentétben.

A kamera és a mikrofon hiányával a rendszer használhatatlanná válik, ugyanis a két input ezen két eszközön érkezik.

2.5. Következtetés

A különböző megközelítéseket átfésülve és a megszorítások tudatában olyan következtetéseket vonhatunk le, ami segít egy olyan rendszert kivitelezni, ami magában hordozza a már meglévő asszisztensek előnyeit, és megpróbálja kiszűrni azok hátrányait.

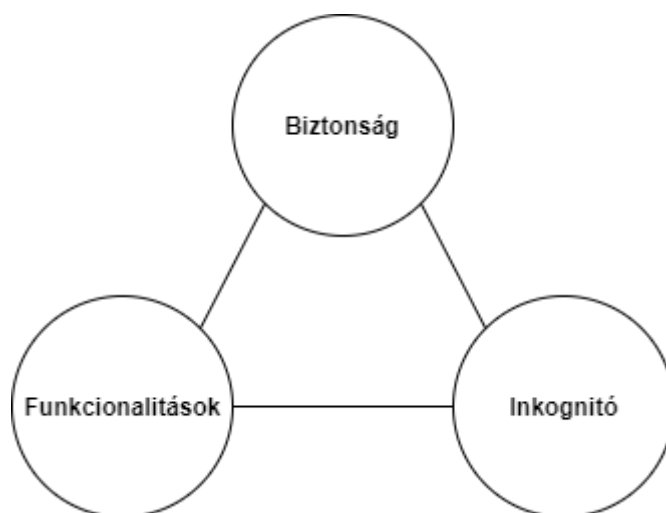
Elmondható, hogy egy általános és jól működő virtuális asszisztens alapjai a következők:

- **Eszköz:** a rendszer egy eszközön fut, fontos, hogy a felhasználó számítógépe megbírja a rendszer terhelését.
- **Azonosítás:** a virtuális asszisztensnek tudnia kell, hogy éppen a gazdája beszél-e hozzá, ezáltal szükségünk van valamiféle azonosításra, lehet az egy jelszó, hang alapú, vagy kép alapú azonosítás is. Szükség van egy azonosításra, aminek egyedinek és nehezen kizárhatóan kell lennie. A képalapú azonosítás a dolgozat másik része.
- **Biztonság:** amiért az azonosítás annyira fontos a virtuális asszisztensek szempontjából az nem más, mint a biztonság, pontosabban adatbiztonság. Egy tömör dologhoz hozzáférő asszisztens, ha rossz kezekbe kerül maradandó károkat tud okozni.

- **Gyorsaság:** a rohanó világunkban senki nem szeretne várni semmire, ezért a gyorsaság és a hatékonyság egy nagyon fontos funkció esetünkben. A rendszer fő funkcionálisa kiszolgálni a felhasználót elfogadható időkorlátok között.

A rendszer fő jellemzője a gyorsaság, egyszerűség és hatékonyság kell legyen a megbízhatóság mellett, ezeket figyelembe véve, valamint a szükséges alapkövetelményeknek megfelelően megalkotható a személyes virtuális asszisztens.

Összességében egyensúlyt kell találni a fent felsorolt alapok között, az alkalmazás egyszerre tökéletesen nem használhatja ki az összes előnyét, néhol a használhatóság és az biztonság a gyorsaság rovására megy. Szükség van áldozatokra, hogy az alkalmazás életre keljen, jelen esetben a biztonság az inkognitóság kárára megy, csak abban az esetben használható biztonságosan az app, ha Sarah tud bizonyos dolgokat a felhasználójáról.



7. Ábra: A háromszöges probléma¹¹

A 7. ábrából tökéletesen kiolvasható a tény, miszerint nem alkothatunk tökéletes rendszert, jelen esetben a funkcionalitások, a biztonság és az inkognitó jelentik a főbb problémát, ugyanis bármelyiket preferálva, előtérbe helyezve a másik kettő rovására megy.

3. Követelmény specifikáció

A szoftver követelmények tudatában tervezzük meg, majd építjük fel a rendszert. A szoftverek tervezése során nehéz megszabni azt, hogy pontosan hogyan kellene működnie a rendszernek, meg hogyan kellene kinéznie a UI-nak, ehhez jönnek segítségül a követelmények. A követelmények tervezése az a folyamat, amikor meghatározzuk és elemezzük a riválisokat, hasonló rendszereket, valamint a megszorításokat, azt követően ellenőrizzük és ledokumentáljuk azokat. A követelmény, mint fogalom nagyon tág jelentéssel bír az IT-ban, a tárgyalását

általában két részre bontják: felhasználói, illetve rendszer követelményekre, ezek tanulmányozására kerül sor a továbbiakban. [5]

3.1.Felhasználói követelmények

A felhasználói követelmények technikai értelemben magas szintű dokumentálás diagrammokkal kiegészítve, amely elsősorban azoknak a személyeknek készül, akik nincsenek tisztában a rendszer kivitelezésével, továbbá nem rendelkeznek részletes háttérinformációval, technikai ismeretekkel a rendszerről. Következtetésképpen elsősorban a felhasználóknak, az ügyfeleknek, illetve ezeket képviselő menedzsereknek készül. A felhasználói követelményekben helyet kapnak azok az információk, amelyek a működést, szolgáltatásokat foglalják magukba több helyen diagrammokkal szemléltetve. [5]

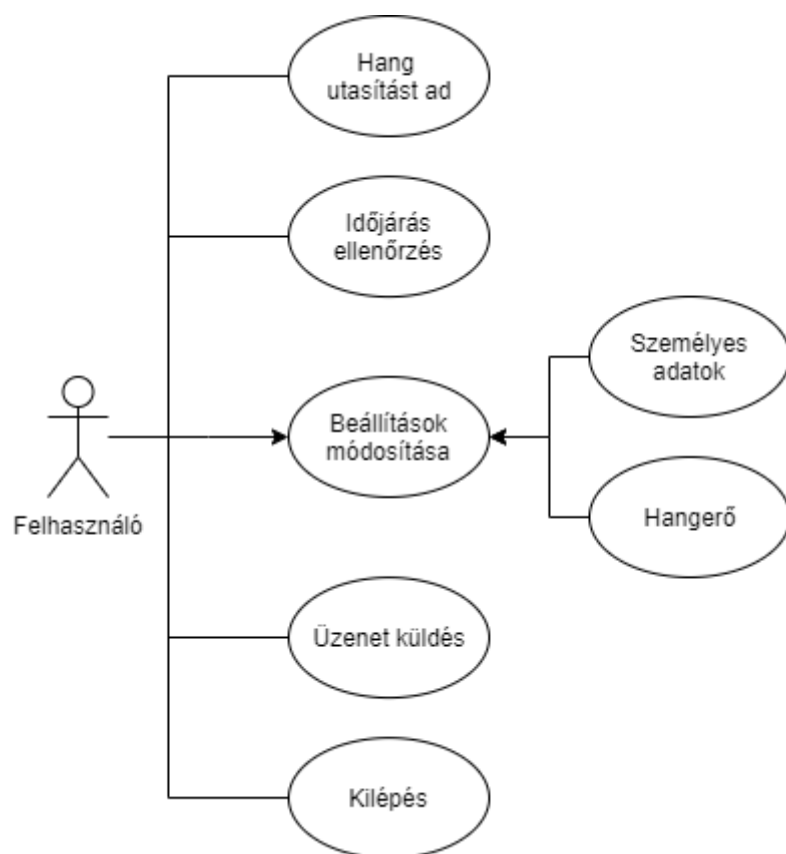
A Sapientia Sarah névre hallgató virtuális asszisztens a mindennapok megkönnyítésére lett kitalálva, továbbá az időnkkel is takarékoskodhatunk, amennyiben használjuk. Az összes funkció a felhasználó előnyeire szolgál. A rendszer által kínált lehetőségek a mindennapi, már-már rutinná vált alapvető műveletek leegyszerűsített elvégzését teszi lehetővé, tehát a legtöbb olyan szolgáltatást tartalmaz, amelyek a felhasználók leggyakoribb problémáit küzdik le. Elég csak arra gondolni, hogy egy átlagfelhasználó milyen műveleteket végez a számítógépén: e-mailokat ír, böngész az interneten, ide érte a Google-t, a Youtube-ot, vagy a Wikipédiát, de lehetnek olyan dolgok is, hogy elindítson egy applikációt, vagy lekérje az aktuális időjárást a világ összes pontjáról. Sarah ezen kéréseket rendre ki tudja szolgálni az egyéb funkcionalitások mellett.

Ahhoz, hogy a felhasználó használni tudja az alkalmazást szüksége van egy számítógépre, egy működő webkamerára és egy működő mikrofonra, internetkapcsolatra.

Az összes ábra egy online weboldalas editor¹¹ használatával készült, amely egyszerű elkészítést, illetve módosítást tett lehetővé.

A rendszer „use-case” diagramját a 7-es ábrán lehet szemügyre venni, ezen megtalálhatók a használati esetek a rendszer szempontjából. Esetünkben a diagram célja szemléltetni a felhasználó szemszögéből a program működését úgy, hogy az könnyen áttekinthető legyen.

¹¹ <https://www.diagrameditor.com/>



8. Ábra: Az alkalmazás Használat-Eset diagramja

A 8. ábrán fellelhető a rendszer Használat-Eset diagrammja továbbiakban sor kerül a fontosabb használat-esetek elemzésére, kiértékelésére, a magasszintű táblázatokban való kiértékeléshez egy sablont¹² használtam:

Hang utasítást ad.	
Szereplő:	A felhasználó, akinek a gépén fut az alkalmazás.
Leírás:	A Virtuális Asszisztenshez kapcsolódik, pontosabban azon részéhez, ahol a felhasználó hang utasítást ad a rendszernek.
Előfeltételek:	<ul style="list-style-type: none"> - Mikrofon megléte. - Engedélyezett mikrofon modul.
Utófeltételek:	<ul style="list-style-type: none"> - Programon belüli navigáció.
Folyamat:	<ul style="list-style-type: none"> - Indítást követően a főképernyőn megjelenő status mező zöldre válásával a rendszer hallgat bennünket, majd a feldolgozást követően végrehajtja a kérést és visszajelzést is küld számunkra, ez nevezhető a sikeres folyamatnak.

¹² Use Case templates: <https://templatelab.com/use-case-templates/>

Alternatív folyamat:	<ul style="list-style-type: none"> - Ha abban a pillanatban beszélünk a rendszerhez, amikor a status mező piros, semmi nem fog történni, ugyanis a rendszer ebben az állapotban nem képes kiszolgálni bennünket. - Előfordulhat olyan lehetőség is, hogy a rendszer megérti a parancsot, ám a végrehajtás közben elakad különféle meghatározó külső tényezők hiánya miatt, ilyen például az internetkapcsolat hiánya. - Ha a rendszer nem ismeri fel a felhasználó által kért parancsot különféle kivételek lépnek érvénybe, amelyeket a rendszer tudat a felhasználóval emberi nyelven és status üzenet formájában is.
Kivételek:	<ul style="list-style-type: none"> - Ha nem tud megnyitni egy weboldalt a következő mondattal jelzi a felhasználó felé: „I can't see it”. - Ha nem tudja megnézni egy márka jelenlegi tőzsdei értékét a követő mondattal jelzi a felhasználó felé: „oops, something went wrong”. - Amennyiben a felhasználó által kért böngésző nincs telepítve, vagy nem találja az indító fájlt az alapértelmezett Internet Explorert nyitja meg. - Ha probléma adódik az operációs rendszer beállításainak megnyitása közben a Vezérlőpultot tárja elénk. - Powershell helyett a CMD startol, amennyiben az előbbi valamilyen hiba folytán nem tud a képernyőre vetítődni. - Globálisan, ha egy alkalmazást nem tud elénk tárni Sarah azt feltételezi, hogy nincs telepítve, így a következő utasítást mondja nekünk: "Seems like the application is not installed" - Amikor nem tudja lekérni az időjárást az API-ból „Error” üzenetet tár elénk. - A Youtube-on, Google-ban, vagy akár a Wikipédián való keresés esetén, ha valami befuccsol a "Please try again." kivétellel találjuk szembe magunkat. - A navigáció amennyiben nem teljesül az „I couldn't find the

	<p>place” visszajelzést küldi a rendszer a felhasználó felé.</p> <ul style="list-style-type: none"> - Ha az e-mail küldő részénél probléma adódik az „I cant send your e-mail!” visszajelzés érkezik felénk.
Követelmények:	<ul style="list-style-type: none"> - Pontosság - Gyorsaság

Személyes adatok módosítása.	
Szereplő:	A felhasználó, akinek a gépén fut az alkalmazás.
Leírás:	A Virtuális Asszisztenshez kapcsolódik, pontosabban azon részéhez, ahol a felhasználó megadja a személyes adatait a rendszernek
Előfeltételek:	<ul style="list-style-type: none"> - A felhasználó rendelkezik a kért személyes adatokkal. - Billentyűzet, vagy virtuális billentyűzet megléte.
Utófeltételek:	<ul style="list-style-type: none"> - Programon belüli navigáció. - A felhasználó elfogadja-e a program által kért adatvédelmi irányelveket. - A rendszer elfogadja-e a felhasználó által beírt adatokat, amelyeknek különféle követelményeknek kell megfelelnie.
Folyamat:	<ul style="list-style-type: none"> - Az applikáció elindításával megjelenő főoldal „Settings” gombjára kattintva megjelenik a legördülő listában egy „Personal” gomb, ez megnyitja az oldalt, ahol frissíthetjük az adatainkat. - A különféle mezők kitöltését követően az OK gombra kattintva a rendszer az ablak aljában kiírja zöld színnel, hogy frissítve lettek az adataink.
Alternatív folyamat:	<ul style="list-style-type: none"> - Amennyiben probléma adódik a mezők kitöltésénél életbe lép és látható lesz egy úgynevezett „Warning Label”, amely piros színnel tárja elé a problémát, ami nem felel meg a rendszer elvárásainak. - Ha a dobozok kitöltését követően a Back gombra kattintunk elvesznek a frissített adatok, nem kerülnek elmentésre és bezárja az ablakot a rendszer. - Az adatvédelmi szabályok szövegre kattintva elolvashatjuk

	azokat, még mielőtt elfogadnánk. Ez az ablak elérhető a „Help” menüpont alatt is.
Kivételek:	<ul style="list-style-type: none"> - Az úgynevezett „Warning Label” a „Please fill in all fields” üzenetet tárja elénk, amennyiben egyetlen bemenetet váró doboz is érintetlen maradt. - Ha a megadott születési év meghaladja az éppen aktuális évet, akkor kiírja, hogy „Year should be less than <aktuális év>” - A teljes névnek legalább 2 szóból kell állnia, ellenkező esetben olyan figyelmeztetést kapunk, hogy „I need your full name!” - Az e-mail cím esetében a formának valid e-mailcímhez kell hasonlítania. Ellenkező esetben az „I need a valid email address!” üzenet tárul elénk. - „Error” visszajelzést kapunk, amennyiben nem sikerül valami oknál fogva behelyezni az adatokat a konfigurációs fájlba.
Követelmények:	<ul style="list-style-type: none"> - Pontosság - Biztonság

Hangerő módosítás	
Szereplő:	A felhasználó, akinek a gépén fut az alkalmazás.
Leírás:	A Virtuális Asszisztenshez kapcsolódik, a felhasználó szempontjából írja le hangerő módosítási lehetőséget.
Előfeltételek:	<ul style="list-style-type: none"> - Hangszórók, vagy hangkibocsátó modul(ok) megléte - Szoftveres lehetőség ezen modulokat módosítani.
Utófeltételek:	-
Folyamat:	<ul style="list-style-type: none"> - A módosításhoz a „Settings” fül „Volume” ablakát kell megnyitni. - A megnyitást követően egy 0-tól 100-ig terjedő csúszkán állíthatjuk be a kívánt rendszer hangerőt, majd ezt a „Confirm” gombbal véglegesíthetjük.
Alternatív folyamat:	<ul style="list-style-type: none"> - A „Cancel” gombra kattintva a csúszkán megadott érték elvész és az ablak bezáródik.

Kivételek:	-
Követelmények:	<ul style="list-style-type: none"> - Pontosság - Gyorsaság - Biztonság

Időjárás ellenőrzés	
Szereplő:	A felhasználó, akinek a gépén fut az alkalmazás.
Leírás:	A Virtuális Asszisztenshez kapcsolódik, a felhasználó szempontjából írja le a lekért időjárás megtekintésének lehetőségét.
Előfeltételek:	<ul style="list-style-type: none"> - Érvényes API kulcs megléte. - Friss adatokhoz friss lekérés.
Utófeltételek:	<ul style="list-style-type: none"> - Utolsó lekért adatot is meg tudunk jeleníteni. - A hangfelismerő rendszerben a „weather” illetve az „in” kulcsszavak megléte lekéréskor, az utóbbi után a település neve.
Folyamat:	<ul style="list-style-type: none"> - A hangfelismerő rendszerben az API lekéréshez „weather” és az „in” kulcsszót követően meg kell jelennie a település névnek. - Feldolgozásra kerül a parancs, majd a „Help” menüpont alatt a „Weather” gombra kattintva előjön az ablak, az adatok megjelenítéséhez meg kell nyomnunk a „Show Me” gombot az ablak jobb alsó sarkában. - Egy szöveges állomány jelzi a UI-on azt, hogy mikor volt az utolsó API lekérés. - Miután megtörtént az API lekérés, és frissítve lettek az adatok az egyébként rejtjelezett konfigurációs fájlban, Sarah természetesen jelzi ezt felénk a „Please check the Weather GUI.” mondattal. - Az ablak bezáráshoz szükséges a „Back” gombra kattintani.
Alternatív folyamat:	<ul style="list-style-type: none"> - A „Back” gombra nem íródnak ki az adatok.
Kivételek:	<ul style="list-style-type: none"> - Ha hiba történik az API lekérés közben az „Error” hibaüzenettel találjuk szembe magunkat.

Követelmények:	<ul style="list-style-type: none"> - Pontosság - Gyorsaság - API kulcs
----------------	---

E-mail küldés	
Szereplő:	A felhasználó, akinek a gépén fut az alkalmazás.
Leírás:	A Virtuális Asszisztenshez kapcsolódik, a felhasználó szempontjából írja le azt, hogy hogyan tud E-mailt küldeni egyszerűen Sarah segítségével.
Előfeltételek:	<ul style="list-style-type: none"> - Érvényes e-mail cím és annak a valid jelszavának megadása a regisztrációnál, vagy később az adatmódosítási lehetőségnél. - Amennyiben a fiók Google fiók engedélyezni kell a hozzáférést a kevésbé biztonságos alkalmazásokhoz. - Valid e-mail cím a fogadónak.
Utófeltételek:	<ul style="list-style-type: none"> - A rendszer ellenőrzi, hogy a standardnak megfeleljen az e-mail cím.
Folyamat:	<ul style="list-style-type: none"> - A „File” menüponton belül az „E-mail” lehetőségre kattintva megjelenik az ablak, amely a küldő E-mail címét, illetve a szövegdobozt tartalmazza, ahová az üzenet kerül. - A „Send” gombra kattintva az E-mail elküldésre kerül.
Alternatív folyamat:	<ul style="list-style-type: none"> - Amennyiben ki szeretnénk törölni könnyedén az alapértelmezett sablont ezt könnyen megtehetjük a „Clear” gombbal, ami az üzenet dobozban levő összes karaktert törli. - A „Back” gombra kattintva nem kerül elküldésre az üzenet, hiába töltöttük ki a mezőket, azok az információk eltűnnek és az ablak bezárásra kerül.
Kivételek:	<ul style="list-style-type: none"> - Ha hiba történik az E-mail küldése közben az „I can't send your e-mail!” üzenet érkezik az úgynevezett „Warning Label”-en keresztül a felhasználó felé, amely figyelemfelkeltő piros színű. - Ellenkező esetben, amennyiben az e-mail küldés sikeres volt a „Your e-mail is sent!” üzenet érkezik zöld színnel ugyanezen

	<p>a labelen keresztül.</p> <ul style="list-style-type: none"> - Ha az üzenet mezőt üresen hagyjuk a „*WARNING* - Message box is empty!” üzenet érkezik, ha az E-mail mezőt ehhez hasonlóan ugyanúgy piros színnel. - Ha az e-mail formája nem felel meg a standardnak a „*WARNING* - I need a valid email address!” figyelmeztetéssel találjuk szembe magunkat piros színnel.
Követelmények:	<ul style="list-style-type: none"> - Pontosság - Gyorsaság - Megbízhatóság

3.2. Rendszerkövetelmények

A rendszerkövetelmények tartalmazzák a szoftver funkciók és megszorítások leírását és azt, hogy mit kell leprogramozni a rendszerbe. A rendszerkövetelményeknek két típusát különböztetjük meg, amelyekről szó lesz a következőkben: funkcionális és nem-funkcionális követelmények. [5]

3.2.1. Funkcionális követelmények

A funkcionális követelmények tartalmazzák a leírást arról, hogy a rendszernek milyen funkciókkal kell rendelkeznie és ezek a funkciók hogyan kellene működjenek, tehát az inputokra érkező outputokat írja le. [5]

A rendszer hang utasítást vár a felhasználotól a főképernyőn (11. Ábra) elhelyezett Start gomb megnyomását követően egészen addig, amíg a Stop gomb meg nem nyomódik – ebben az esetben nem áll meg a hallgatás, csak szünetel. A rendszer elindításakor az üdvözölni fogja a felhasználóját napszaktól függően. Feltételezzük, hogy a felhasználó másfél méteren belül helyezkedik el a számítógéphez csatolt, vagy abba beépített mikrofontól: ez egy ajánlott távolság, hogy Sarah tökéletesen fel tudja dolgozni a hanganyagot. Emellett ajánlott a háttérzaj kiszűrése amennyire lehet, bár Sarah is rendelkezik szűrő funkcióval. Az alkalmazás folyamatosan fogadja hangbementet a fő szálon, amit majd egy hátsó szállnak ad feldolgozási munkasorba. Az alkalmazás által használt több szálás megoldás lendít a gyorsaságon úgy felhasználói felület, mint beszédfelismerés és végrehajtás terén. Mindez abban az esetben

szemaforos megközelítéssel gátolva van, amikor Sarah úgymond beszél, a rendszer lejátssza a betáplált hangfájlt, akkor a hallgatás leblokkol.

A feldolgozás során a hanganyag szöveggé alakítva végig megy az ezért felelős részeken, ahol egyezést észlel az elvárt bemenet és a tényleges bemenet között oda belépik és elvégzi a végrehajtást. Ezt követően impulzust kap a felhasználó erről két féle módon: Sarah női hanggal értesíti a felhasználót arról, hogy a munkát elvégezte, továbbá gondolva arra az élethelyzetre is, amikor az eszköz néma üzemmódban van a főképernyőn megjelenő szövegdobozba is kiírja az elmondott üzenetet. Amennyiben nem értette az általunk elmondott parancsot arról is visszajelzést kapunk. Ezen feedback-ek viszonylag gyorsan érkeznek, általában megközelítőleg 2 másodpercen belül, természetesen ez függ a számítógép leterheltségétől, illetve attól is, hogy milyen komplex a kért parancs végrehajtása.

A hang bemenet mellett a hang kimenet is nagyon fontos, ugyanis ha éppen a felhasználó munkát végez és nem olvassa a Sarah féle visszajelzéseket akkor azt hallgatnia kell. Ezen hangkimenet erősségét is lehet állítani alkalmazáson belül a beállítások menüpont alatt a hangerő beállításokra kattintva, ahol egy új GUI-t fedezhetünk fel, amelyen egy sáv található, ahol a hangerőt tudjuk szabályozni, majd ezt végrehajthatjuk, illetve elvethetjük.

Az alkalmazás lehetőséget ad arra, hogy az adatainkat meg tudjuk változtatni, erre is egy külön GUI szolgál a beállítások menüpont alatt.

Amennyiben végleg meg akarjuk szakítani az applikáció működését és nem csak a hallgatást szeretnénk szüneteltetni erre is lehetőségünk adódik az Exit gomb lenyomásával, továbbá gyorsbillentyű kombináció is van beépítve a rendszerbe, az ALT+F4 egyszeri lenyomásával kilépünk az alkalmazásból, de amennyiben használjuk a „Bye” szócskát, vagy egyéb elköszönésre alkalmas szót, mondatot Sarah elköszön tőlünk és alvó üzemmódba lép.

A funkcionális követelmények közé sorolható az, hogy az asszisztens képes legyen tájékoztatni minket az aktuális időpontról, továbbá a dátumról. Sarah esetében erre nagy hangsúlyt fektettem, kérdésre válaszol, továbbá a boxon kívül ugyancsak a főképernyőn látható megjelenítve a napi dátum, továbbá másodpercre pontosan az aktuális idő. Az idő mellett a felhasználókat az aktuális időjárás is érdekelheti, Sarah erre is megoldást nyújt. A világ bármely települését eléri egy API-n keresztül, így folyamatosan friss adatokat közölhet. A rendszer élénk tár egy külön oldalt ahol megtalálhatók az adott településről legfontosabbnak vélt adatok, ilyen például a település neve, a hőmérséklet Celsius fokban, a hőérzet ugyancsak ebben a mértékegységben, vagy a szélerősség. A GUI alján folyamatosan megfigyelhetjük, hogy mennyire friss az adat, mikori a legutolsó lekérés.

A virtuális asszisztensek egyik legalapvetőbb feladata a készüléken levő alkalmazások megnyitása, mindez Sarah-ba is egy betáplált, elérhető funkció. Nem titok, az alkalmazás leginkább Windows operációs rendszerre van öltöztetve, így az alapértelmezett programok elindítása nem jelent gondot. Egy felmérést készítettem kérdőív formájában a rendszerről, és az is a kérdések között szerepelt, hogy a felhasználók általában milyen alkalmazásokat futtatnak, használnak a számítógépükön. A kérdőívet közel 88-an kitöltötték a kiértékelést megelőzően, tömérdek applikáció tippet kaptam, amelyet a user szívesen használ és amelyet a rendszernek ismernie kell. Ilyen például a Zoom, a Facebook Messenger, vagy akár a Steam. A kérdőívet kitöltők 59.1%-a szerint nagyon fontos az, hogy alkalmazásokat is tudjon megnyitni az asszisztens, egyébként ez volt a legkelendőbb a választási lehetőségek közül, ezt követte az időjárásjelentés ismertetése (52.3%), majd az e-mail küldő funkció, de a 18.2% szerint a meglévő funkciók is elegendők. Itt lehetőség nyílt a felhasználóknak személyes véleményt is megformálni, azonban nem jött érdemleges, kivitelezhető ötlet.

Különböző weboldalak megnyitása mellett külön a Google-n, a Youtuben és a Wikipédián is tud keresni kulcsszavakra, kulcsmondatokra. A kérdőívet kitöltő 88 személy magas százalékát érdekelte ez a funkció, így implementálásra került.

A potenciális felhasználók arról, is visszajelzéseket adtak, hogy szívesen kérdeznék meg a számítógépükön futó Sarah-t indulás előtt arról, hogy navigálja el őket egyik városból a másikba. A rendszer erre is megoldást nyújt a Google Maps megnyitásával, ha azt is elmondjuk neki, hogy honnan és hová szeretnék utazni azonnal kiválasztja nekünk a legrövidebb útvonalat.

3.2.2. Nem-funkcionális követelmények

A nem-funkcionális követelmények tartalmazzák a rendszer megszorításokat és a működési feltételeket. Ezen követelményekben a fejlesztésre vonatkozó leírásokat is megtaláljuk. Ez a rész drasztikusabb az előzőnél, ugyanis egy rosszul, vagy egyáltalán nem működő rendszert kapunk abban az esetben, amennyiben a feltételek nem teljesülnek. [5]

Manapság kulcsfontosságú elvárás a felhasználói szemszögből az, hogy az alkalmazás válaszüzeje a lehető legrövidebb legyen, az hatékony és megbízható legyen, de a hordozhatóság és a felhasználhatóság is igencsak mindennapossá vált, ezek hatványozottan igazak egy virtuális személyi asszisztens esetében. Mindazon előnyök mellett, amelyet az alkalmazás nyújt a felhasználói réteg számára a megfelelő működés elérése érdekében néhány alapvető követelményre szükség van.

Fontos a hordozhatóság, mint ahogyan azt korábban leírtam, tehát eszköz nélkül nem használható a program, egy laptop, vagy asztali számítógép szükséges ahhoz, hogy a felhasználó igénybe vehesse Sarah kvalitásait. Az alkalmazás futtatása Linux, Windows és macOS operációs rendszereken lehetséges, ezek is nélkülözhetetlen kellékek.

A használt könyvtárak egy része publikus felhő alapú szolgáltatást vesz igénybe, amelyhez elengedhetetlen az, hogy az eszköz, ahol az alkalmazás fut csatlakozva legyen a világháléhoz.

A felhasználói szemszögből nagyon fontos az, hogy a rendszer a leghatékonyabb legyen, jelen esetben fontos az, hogy a válaszidő a lehető legkevesebb legyen. Ehhez természetesen elengedhetetlen egy kvalitásban gazdag számítógép: minél erősebb gépen futtatjuk az alkalmazást, annak a válaszideje annál gyorsabb, az imént említett felhő alapú könyvtárak miatt itt az internet sebessége is közrejátszik.

Mindezek mellett szükség van egy legfeljebb 200 MB-os szabad tárhelyre mert a debug fájlban rögzítve van minden, és az folyamatosan terjeszkedik, továbbá némi RAM-ra, minél többel rendelkezik az eszköz, annál gördülékenyebben működik az alkalmazás.

Ahhoz, hogy az alkalmazás működni tudjon kell egy beépített, vagy egy csatolt mikrofon, ami végrehajtásra váró hang bemenetet rögzít, az alkalmazás futása közben folyamatosan egy hátsó szálon.

A rendszer e-mailt küldő részéhez elengedhetetlen egy valid e-mail cím és egy ehhez tartozó jelszó. Ezek nélkül lehetetlenné válik az, hogy az alkalmazás ezen funkcióját igénybe vegye bárki is. Mindezek mellett ahhoz, hogy a funkció működni tudjon ki kell legyen kapcsolva az érintett accounton történő fokozott biztonsági funkció, hiszen egyébként más, „idegen” készülékről kell hozzáférnünk az e-mail fiókhoz.

Az adatbiztonság az egyik legkényesebb dolog a szoftveriparban, tömérdek hátulütője van a témakörnek, minden tervezésnél az egyik alapkőnek kell lennie. A Sarah-t igénybe vevő felhasználóknak tudniuk kell, hogy a regisztrációnál megadott adatok kriptálva vannak elmentve, így senki nem férhet hozzá a személyes adatokhoz. Amikor a rendszer a személyes adatainkból nyer információt, visszafejti a config fájl tartalmát egy rövid időre szöveges formátummá, majd újra rejtjelezi azt, mindezt a másodperc töredéke alatt.

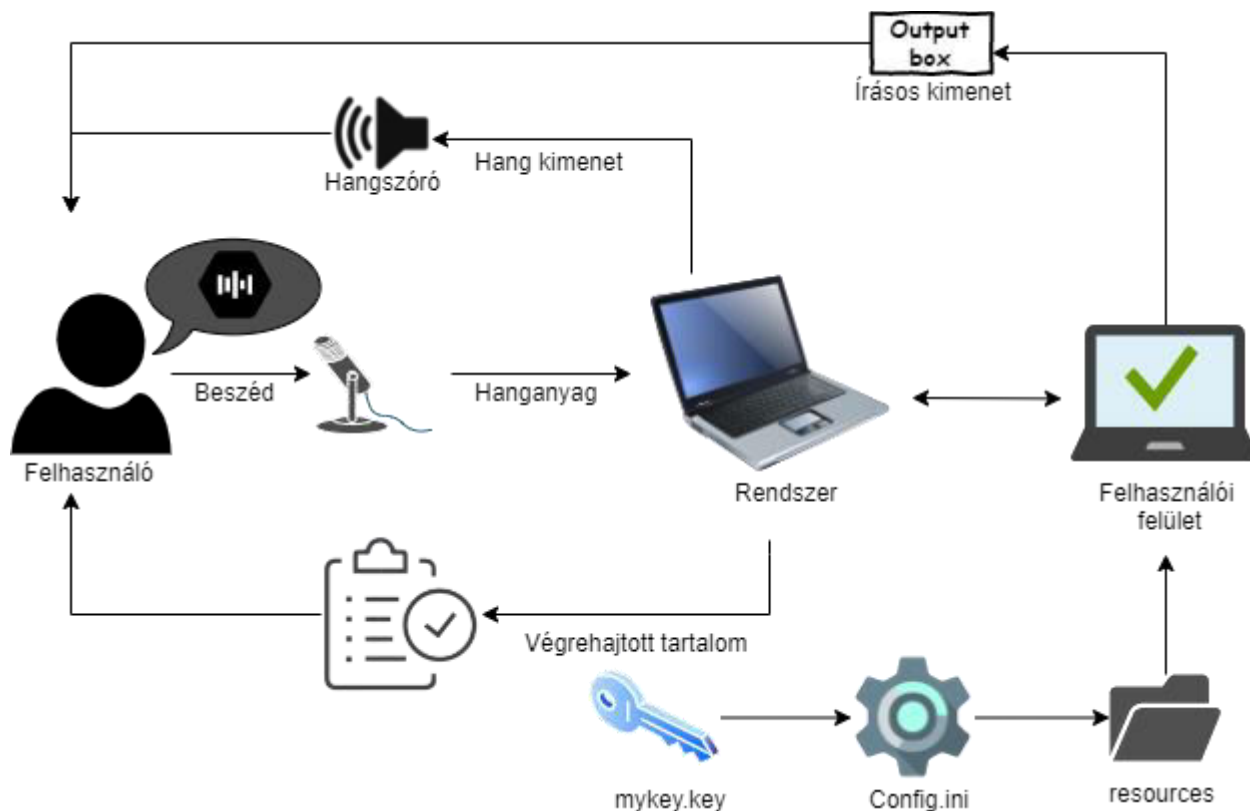
A nem-funkcionális követelmények közé sorolható a szervezés is. A konzultálás heti rendszerességgel megvalósult, amikor kimaradt, akkor pótlásra került. Ezen alkalmakkor prezentálás történt a részemről, Dr. Szántó Zoltán ezeket követően egyengette folyamatosan az utamat és az új hétre új kihívásokat, kutatásokat adott számomra.

A szoftverfejlesztésben alapvető a verziókezelő rendszer jelenléte, enélkül a kód változtatások követése elképzelhetetlen és lehetetlen lenne a vezetők és a fejlesztők által. Személy szerint a tapasztalataimból kiindulva a GitHub¹³ verziókezelő rendszert használtam.

¹³ Github: https://github.com/magyarosiR/Virtual_Assistant/

4. A rendszer architektúrája

A rendszer architektúrája úgy lett megtervezve, hogy azt könnyen lehessen implementálni, illetve kibővíthető legyen egyéb modulokkal kompromisszumok nélkül.



9. Ábra: A rendszer architektúrája¹¹

A 9. ábra a rendszer architektúráját mutatja be, ahol megtalálható az összes elem, összetevő úgy szoftver, mint hardver részről.

Az egész a felhasználóból indul ki, amely a számítógéphez kapcsolt, vagy abba épített mikrofonba beszél, mindez hanganyagot küld a szisztémának, ami az eszközön fut. A backend kölcsönhatásban van a felhasználói felülettel, magával a GUI-val, amit a felhasználó is lát az eszköz képernyőjén. A rendszer hanganyagot küld vissza a felhasználónak, egyidőben ugyanezt írásos formátumban is megteszi a felhasználói felületen megtalálható kimeneti szövegdobozban. A végrehajtott tartalmat is a rendszer szolgáltatja a felhasználó felé, amely általában háttérbe szorítja a GUI-t, mint vizuális felületet az eszköz kijelzőjén. A „resources” mappa szolgáltatja az alkalmazás felhasználói felületének az iconokat, illetve itt található meg a logók is, tehát Sarah innen nyeri a nyersanyagokat, amelyeket a felhasználó elé tár.

A felhasználó az a személy, aki a szolgáltatást igénybe veszi, használja. A felhasználó esetében az alsó korhatár nincs megszabva, azonban leginkább a felnőtt korosztályt céloztam

meg a tervezésnél, ugyanis Sarah első számú feladata a munkát könnyíti meg valamilyen szinten. A felhasználó esetében alapkövetelmény az, hogy kiforrott beszédképességgel rendelkezzen, továbbá minimális angol tudásra is szükség van ahhoz, hogy felhőtlenül élvezze a számítógép előtt ülő a rendszer előnyeit és a kínált lehetőségeket.

A rendszer egy asztali számítógépen, vagy laptopon fut. Az alap géphez egyéb architekturális elem nem kell tartozzon a megszokott, már alapnak számító egér és mikrofonon kívül. A gépen futó szisztéma viszonylag kevés helyet foglal a memóriából, viszont a többszörös megközelítés miatt nagy a processzorigénye. A géptől függ az alkalmazás hatékonysága. A rendszer kölcsönhatásban van a felhasználói felülettel.

A felhasználói felület több ablakból áll, ezek mérete általában megközelítőleg 800x500 pixel, de vannak kisebb és nagyobb formák is. Az ablakok mérete rögzítve van, nem méretezhető. Indításkor a MainScreen nevű oldallal találjuk szembe magunkat, lényegében itt lehet kezelni az egész rendszert, a hangvezérlést és egyebet. A különböző komponensek különböző szerepeket látnak el. A mi felületünk lényege is ugyan az, mint a mobilos, webes, vagy egyéb alkalmazások esetében: a felhasználói ezen keresztül kommunikál a rendszerrel, továbbá a visszajelzéseket is itt olvashatja, láthatja, a hang visszajelzések kivételével. A felhasználói felület egy a mai trendeknek megfelelő sötét, úgynevezett „Dark” témát kapott, kevés, ám visszatérő színekkel kombinálva, amelyből nem maradhat ki a Sapientia logójában fellelhető sötét, amolyan már fakóba átmenő zöld sem.

A „resources” állomány tartalmazza az összes nyersanyagot, amelyre szüksége van a felhasználói felületnek. A mappában fellelhető az összes használt ikon és a logó, továbbá a különféle fejlécek és stíluslapok. Ezek mellett a felület egészében használt „ModernSans-Light” betűtípust is innen találják meg az azt használó komponensek. Ezen három rész külön mappákban el van határolva egymástól.

A „config.ini” fájlba olyan adatok vannak elmentve, mint a felhasználó kényes, személyes adatai, az időjárás lekérő API eredményei, valamint az e-mail küldéshez szükséges adatok. A fájl teljes tartalma rejtjelezve van, így nem kell attól tartani, hogy bárki is hozzáférhet ezen információkhoz. A rejtjelezéshez szükséges a „mykey.key” file, amiben a kulcs van eltárolva, amely segítségével a rendszer vissza tudja fejteni arra az időre a fájlt, amíg kinyeri belőle a kellő adatokat. Amint megkapta azokat Sarah azonnal újra kriptálja a fájlt. Ezen fájl tartalma akkor változik, amikor módosítjuk, frissítjük a személyes adatainkat, akkor mindig új kulcsot generál a rendszer.

A végrehajtott tartalom elsősorban nem a UI részen fellelhető, mint egyéb alkalmazások esetében, ugyanis a virtuális asszisztenseket előszeretettel használják interneten való

böngészésre, különféle alkalmazások megnyitására stb. Néhol a végrehajtott tartalom a hangszórón keresztül érkezik hangkimenetként, amolyan válaszként a rendszertől.

A hang kimenet a számítógép hangszóróján érkezik felhasználóhoz vissza, a rendszer úgy lett konfigurálva, hogy minél több visszajelzést küldjön a felhasználónak, általában hanggal, mintsem képpel, ugyanis az alkalmazás a tervek szerint javarészt a háttérben fog futni, így nem kell folyton előtérbe helyezni azért, hogy megnézhessük a visszajelzést. A legtöbb munkavégzést követően hozzánk szól Sarah valamilyen formában. Ezen visszajelzéseket az Output Boxban is megnézhetjük, írásos formátumban.

4.1.A felhasználó

A rendszer tulajdonképpen bárki számára elérhető és használható, nincs korhoz, nemhez, vagy iskolai végzettséghez kötve. A célközönség a számítógépen dolgozók, vélhetően számukra a legjövedelmezőbb Sarah használata, aki könnyen elvégezhető, rutinszerű kattintásokat spórol meg. A felhasználó telepíti az alkalmazást, majd beléphet a rendszerbe. Regisztrálástól függően a hangbemenetet nem különbözteti meg a rendszer, így bárki mondhatja neki a parancsokat, aki hozzáfér a számítógéphez.

A projekt másik részéhez – amely önmagában egy másik államvizsga dolgozat – tartozik egy regisztrációs lap, valamint egy kamerás modul megléte, amely előre definiálja a számítógép előtt ülőt, arcfelismeréssel. Amennyiben átment ezen a teszten a felhasználó és valóban a regisztrált arcot találja meg a rendszer beléphet, ám azt követően az újraindításig nincs ellenőrizve, hogy ki ül a rendszer másik oldalán, a mikrofon mögött.

4.2.A felhasználói felület

A felhasználó szemszögéből az egyik legfontosabb nézet a felhasználhatóság, tehát fontos egy könnyen átlátható és kezelhető felhasználói felület jelenléte. Hiába tökéletes a kommunikáció és az asszisztens által nyújtott tökéletes szolgáltatás, felhasználói felület hiányában gyakorlatilag használhatatlanná válik az átlagfelhasználók számára mindez. A felhasználói felület egyelőre egy asztali alkalmazást foglal magába.

A backend miatt a technológia adott volt, a felhasználói felület többek között ennek hatására PyQt5-ben íródott, ugyanis eléggé korszerűnek és komplexnek éreztem ahhoz, hogy meg tudjak alkotni egy Sarah-hoz hasonló alkalmazás grafikus, szemmel látható részét. A Python vélhetően az egyik legkönnyebben tanulható és a legszebb szkriptnyelv széles körben, ehhez vélhetően a Qt nyújt a legegyszerűbb fejlesztői platform kreálási lehetőséget. A PyQt a Qt

kötélékébe tartozik, amely fejlesztése elsősorban C++ nyelven volt elérhető. A Qt lényegében egy keretrendszer, amelyben GUI-kat lehet létrehozni és módosítani. A Qt nagyon népszerű a C++ szoftverfejlesztők körében, de a Python kötése is nagyszerű dobásnak bizonyult, hirtelen leverte az összes riválisát, mára egyeduralkodónak mondható a Python desktop applikációkat fejlesztő lehetőségek körében. A Qt platform dominál a piacon, ezek mellett megtalálható egy ingyenes fejlesztői szoftver, a Qt Creator¹⁴, amely megkönnyíti a dolgom a következőkben - úgy gondoltam. A GUI megalkotása közben nem használtam az adott szoftvert, kódból készült a GUI, azonban a vezérlőpult a Creator segítségével került kigenerálásra. A rendszer kezdetekor a PyQt5 volt a legfrissebb verzió, 2020 december elsején adták ki a PyQt6-ot. [6]

A tervezés elsősorban a keretrendszer és maga a technológia megismerésével kezdődött [6], egy kezdetleges főképernyőt hoztam létre, amelyet összeillesztettem a backend résszel. Ezeket követően több funkció is helyet kapott a főképernyőn, továbbá a design-t is megalkottam, a manapság divatos sötét stílusra esett a választásom. A design implementálása közben nagyon figyeltem arra, hogy ne használjak sok színt, ezért a sötét szürke (rgb(40,40,40)), a világos szürke (rgb(57,57,57)), az úgynevezett Sapiencia zöld – a Sapiencia logójáról¹⁵ vett színkód – (rgb(28,120,74)), fehér és a piros mellett nem használtam egyéb színeket a MainScreen esetében, próbáltam a többi komponensre is ezt a sémát, ezeket a stíluslapokat ráhúzni. Fontosnak találtam, hogy a korábban megalkotott 10. ábrán fellelhető logó, illetve a különféle fejlécek passzoljanak a környezetbe.



10. Ábra: Sarah logója¹⁶

Az előző gondolatmenetet folytatva a színek, illetve a betűtípus az, ami leginkább meghatározza a logó és a GUI közötti kapcsolatot, összhangot, ezt mindenképp össze szerettem volna párosítani. A betűtípusnál a fejlécek megalkotásakor az úgynevezett „Modern Sans”-t használtam, ezt sikerült beiktatnom a rendszerbe is a QtGui QFont metódusának hála.

¹⁴ Qt Creator: <https://www.qt.io/product/development-tools>

¹⁵ Sapiencia logo: <http://www.sapiencia.ro/hu/sajto/arculati-elemek>

¹⁶ Sarah araként szolgáló grafika: https://www.pngitem.com/middle/bobmoR_female-virtual-assistant-icon-hd-png-download/

A font hozzáadását a következőképpen oldottam meg: egy változóba eltároltam az .otf¹⁷ kiterjesztésű fájl elérési útvonalát, mindez a program main részében található meg, tehát indításkor mindig lekérésre kerül a fájl. A *QFontDatabase* osztály információt nyújt az alapul szolgáló, nyersanyagként felhasználható betűtípusokról. Az *applicationFontFamilies* bővíti ezen gyűjteményt az általunk megadott betűtípussal.

A font felhasználása minden esetben egy séma alapján történik. Inicializálunk egy változót, majd szabadon válogathatunk a *QFont* osztály tömérdek lehetősége közül. Betűméretet állíthatunk, dölthetjük a szöveget, félkövér formát is rá szabhatunk, itt méretezhetjük, mindezt ugyanúgy minden komponens esetén.

A Qt alapjáraton, ebből kikövetkeztethetően a PyQt is saját infrastruktúrát biztosít ahhoz, hogy többszörös alkalmazásokat tudjon készíteni a fejlesztő, amelyhez a *QtCore* modulon belül fellelhető *QThread* import szükséges. A *QtCore* modul alapvetően azokat az osztályokat foglalja magába, amelyek nem közvetlenül a GUI-hoz tartoznak, ilyen például az eseményhurok, vagy a signal/slot mechanizmus. Ezek mellett a platformtól független absztrakciók tárhelye is a *QtCore*, például az imént említett szálak, az osztott memória, a reguláris kifejezések és egyéb más dolog is fellelhető.

A PyQt alkalmazások esetében két szálát különböztethetünk meg, a főszálát, illetve a munkásszálát/szálakat. Az alkalmazás futtatásától annak bezárásáig a főszál folyamatosan fut, ezen található maga a grafikus interfész. Amennyiben egy több időt igénybe vevő feladatot szeretnénk a futtatni azt érdemes, sőt kötelező egy hátsó szálra kötni annak érdekében, hogy a felhasználói felület ne fagyjon le. Jelen esetben az egész úgynevezett backend egy munkásszállra van kötve.

A főszál a PyQt alkalmazások esetében GUI-szál néven is ismert, mert az összes widget-et és grafikus komponens-t ő kezeli. A szálát a main() függvényben indítom, a startot lényegében az .exec() függvényhívás idézi elő az előre definiált QApplication objektumon. Ez a szál kezeli a különféle ablakokat is, valamint a gazdagéppel is ő kommunikál.

Alapértelmezetten minden olyan esemény, vagy feladat, amely az alkalmazás fő szálán zajlik, ebbe beleértve a felhasználó eseményeit is szinkronban futnak, tehát ezért nem célszerű egy régóta futó feladatot, vagy sok időt igénybe vevő feladatot a fő szálon futtatni, mert a következő eseménynek addig kell várnia, amíg a valamilyen módon befejeződik. [7]

Fontos megjegyezni azt is, hogy az összes GUI elemet létre kell hoznia és frissítenie is kell azokat. Ugyanakkor végrehajthat egy időben más feladatot egy másik szálon a program, ami

¹⁷ A betűtípus forrása: <https://www.cufonfonts.com/font/modern-sans>

módosíthatja a GUI komponenseket, ebben az esetben ezek fogyasztóként jelennek meg, amelyek információkat várnak az egyéb szálaktól.

```
1 class dateTimerThread(QThread):
2     signal = pyqtSignal(str)
3     def __init__(self, parent = None):
4         QThread.__init__(self, parent)
5
6     def run(self):
7         self.timer = QTimer()
8         while True:
9             time.sleep(1)
10            try:
11                times = QDateTime.currentDateTime()
12                timeDisplay = times.toString('yyyy-MM-dd hh:mm:ss')
13                self.signal.emit(timeDisplay)
14                self.timer.start(1000)
15            except:
16                pass
```

1. Kódrészlet: Egy munkásszál

Az 1-es kódrészlet esetében egy példa van arra az esetre, amikor egy GUI komponens fogyasztó és egy adott hosszú futás idejű szál kimenetét várja. Ezen szál a főképernyőn egy úgynevezett labelt, címkét módosít a pontos dátummal és idővel, másodpercre pontosan. Pontosabban ez a szál nem módosítja direkt módon a főképernyő komponensét, csak egy jelet bocsát ki, itt kerül szóba a *Slot/Signal* mechanizmus.

Az eseménykezelés egy jelből (signal), illetve egy hozzá tartozó részből (slot) áll. A Qt-ban az objektumok képesek maguktól sugározni, signalokat kiadni. Egy Qt objektum akkor jelez, ha történt vele valami fontos, rendkívüli. A slotok olyan speciális tagfüggvények, amelyek képesek hallgatózni, figyelni arra, ha egy másik objektum jelez. A jel akkor kerül elküldésre, amikor valami potenciális érdeklődés lép fel. Ha egy jel egy réshez van csatlakoztatva, az csak akkor kerül meghívásra, ha a jel kisugárzásra került. Ha nincs összekapcsolva a kettő, akkor semmi nem történik, üres programként viselkedik, nem okoz hibát, a signal-t kiváltó esemény lekezeletlen marad. Az esetünkben nem történik hasonló, mindenhol megtalálható a konnekció. A jelet kibocsátó programrész, vagy komponens nem tudja, hogy az általa kisugárzott jel használva lesz-e valaha, de ez nem is az ő felelőssége. [8]

A mellékszálakat a főszálon indítjuk el egy *.start()* metódus meghívásával arra a szálra, amely azelőtt inicializálva lett, majd a slot/signal mechanizmusnak köszönhetően csatlakoztatva egy függvényhez, amely elvégzi a szükséges problémát.

A függvény lényegében módosítja a UI komponenst azon feltételek szerint, amelyek átjöttek a jelek által a munkásszálról. Abban az esetben, ha a munkásszál direkt módon frissíti, vagy módosítja a komponenseket könnyen elveszítheti a különféle stíluselemeit az alkalmazás,

vagy rosszabb esetben kifagyhat az egész. A `qDebug` osztály kimeneti adatfolyamot biztosít az információk hibakereséséhez. A `qDebug` leginkább akkor használandó, amikor a fejlesztőnek ki kell írnia a nyomon követési információkat egy eszközre, a konzolba, vagy éppen egy szöveges állományba.

A UI komponensek különféle osztályokba vannak implementálva. A felhasználói felületek grafikus interfészeinek számától függ a kellendő UI osztályok száma és ugyanez igaz fordítva is.

A főszálon futó fő UI komponens összetétele több részből áll. A `setupUi` függvényben vannak implementálva az úgynevezett Widgetek, komponensek, mint például a szövegdoz, ahová a rendszer az üzenetet küldi, vagy a státusz bár, vagy éppen a gombok. Ezek mellett a különféle konnekciók is itt lelhetők fel, mint például a gombok összekötése az annak megfelelő funkcióval, ezek klikkelésre aktiválódnak, de a menüsávban létrejövő új GUI-k összekötése is itt lelhető fel, a különféle háttérzálak elindítása mellett.

A szálak indítása ki volt vesézve a fejezet 14. paragrafusában, a konnekciók esetében a komponens megnevezését követően eldöntjük, hogy „clicked” vagy „triggered” eseményt várjon, ezt követően jön a csatolás és a metódus paramétereként az érintett függvényt jelöljük meg. A „clicked” eseményt általában gomboknál használjuk, míg a „triggered”-re akkor kerül sor, amikor egy legördülő listából, menüből választunk ki elemeket.

A `retranslateUi` függvényben lényegében az összes műveletnek, menünévnek és komponensnek nevet ad, amit itt megadunk az előre inicializált `_translate` függvény második paraméterének az lesz látható az első paraméterben megadott objektumnév komponensére írva, legyen az cím, vagy szöveg. Ebben a részben adjuk meg a GUI címét is a `setWindowTitle` beépített függvényt használva. A widgetek nevét `setText`, vagy `setTitle` metódussal állítjuk be, használva az előbb leírt `_translate` függvényt.

Az egyéb GUI-k indítása a 2. kódrészletben van szemléltetve, ahol éppen a Help menüpont alatt levő „About Us” ablakot nyitja meg a rendszer kattintásra. Erre a sablonra épül az összes többi GUI kreálás, értelemszerűen kicserélve az adott Ui komponensre az inicializálást.

```
def aboutUs_help(self):
1     try:
2         self.aboutus = Ui_About()
3         self.aboutWindow = QtWidgets.QMainWindow()
4         self.aboutus.setupUi(self.aboutWindow)
5         self.aboutWindow.show()
6     except Exception as e:
7         pass
8
```

2. Kódrészlet: Egy GUI indítása¹³

A gomb funkciók részbe olyan függvények vannak implementálva, amely különféle, előre megtervezett feladatokat, logikákat látnak el.

4.2.1. A forrásanyagok

A korábban kivesézett betűtípust, illetve a PNG formátumú fejléceket, logókat, illetve az ikonokat, amelyek között ICO formátum is megtalálható a resources mappában lehet megtalálni. A rendszer ide menti az aktuális időjárásról az adatokat egy TXT kiterjesztésű szöveges állományba, illetve a logolásnak létrehozott szöveges dokumentum is hasonlóképpen itt frissül.

A különféle logók és fejlécek megalkotására az iPiccy¹⁸ szoftvert használtam, amely áttekinthető és egyszerű kezelni, saját betűtípus és színek is bevihetők. A Sarah arcát nem én alkottam meg, az internetes böngészések során akadtam rá a PNG formátumú képre¹⁹. Egy ablak ikonjának való betöltését *QtGui QIcon* metódusát használva intézhetjük el.

Megkülönböztetünk ikon, illetve nagy logót/kép betöltést. A PNG kiterjesztésű képet egy úgynevezett *QLabel* widgetbe töltöttem be, amely szöveges, vagy képi megjelenítést biztosít. A *setGeometry* metódussal állítottam be a méretet, illetve az elhelyezkedést. Az első paramétere az X, a második az Y koordinátán való elhelyezkedés, a harmadik a tartalom szélességét, illetve az utolsó a magasságát demonstrálja. A *QPixmap* segítségével megjelenítjük a paraméterként megadott képet az ablakban. Az alapértelmezett állapotában hamis *setScaledContents* engedélyezésével a rendelkezésre álló helyet kitöltve méretezte a pixmapot. Végül a *setObjectName* metódussal az elején létrehozott objektumnak véglegesítjük a nevét.

4.2.2. Írásos kimenet

Az architektúrában fellelhető írásos kimenet célja az, hogy szöveges formátumban is küldjön visszajelzést a felhasználónak, egy konverzáció alakul ki a rendszer és a használója között, hasonlóan egy szöveges üzenetben való kommunikáláshoz. A rendszer az úgynevezett Mastert a nevén szólítva teszi be a szövegdobozba, egy kettőspont után megtalálhatjuk azt a szöveget, amit a rendszer észlelt, majd hasonlóképpen a válasz is ilyen formában érkezik, természetesen Sarah előtaggal.

A főoldalon található szövegdoboz, amely egy *QTextBrowser* osztály, ez gazdag szövegmennyiséget képes befogadni. Ez az osztály a *QTextEdit* kiterjesztett változata. Amikor a

¹⁸ <https://ipiccy.com/>

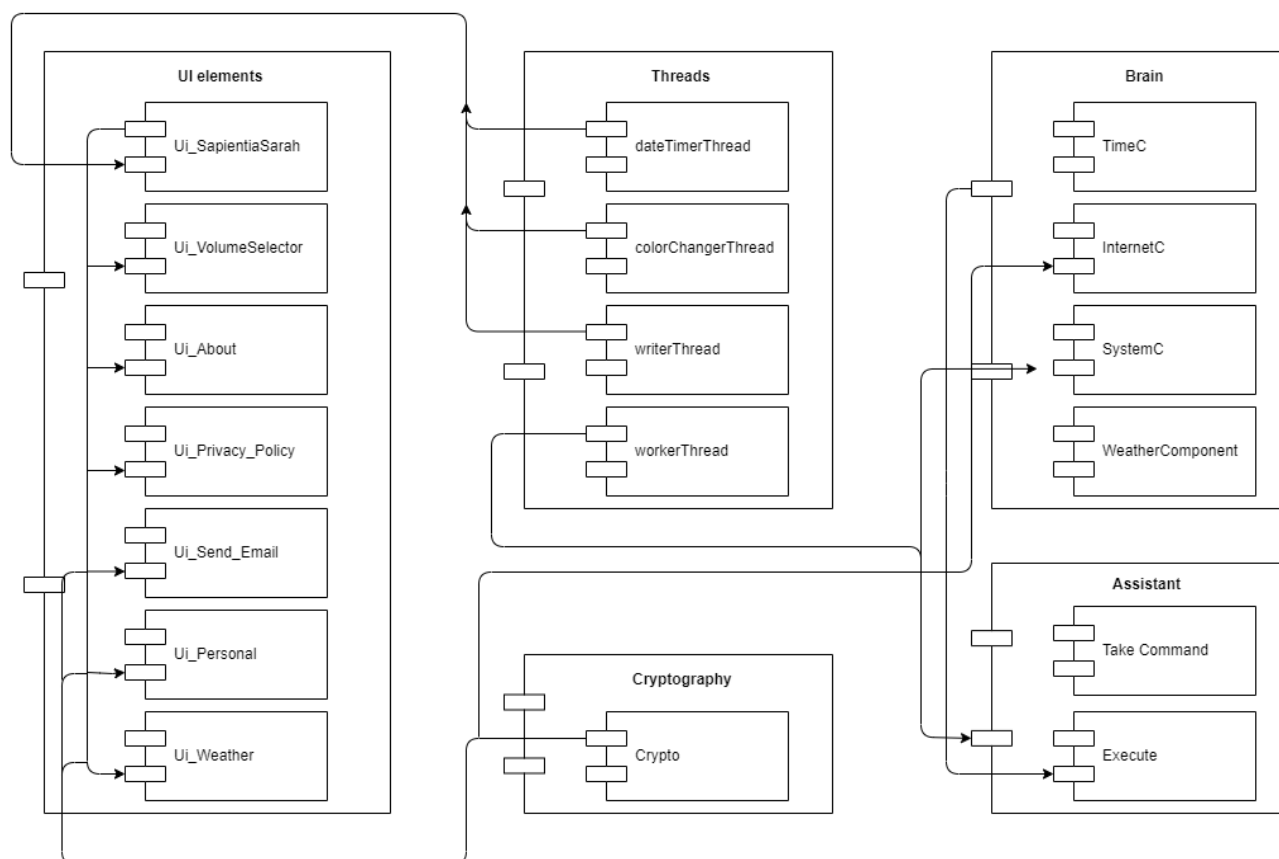
¹⁹ https://www.pngitem.com/middle/bobmoR_female-virtual-assistant-icon-hd-png-download/

rendszer alvó állapotban van a *setText* metódussal beállítottam egy szöveget, amely tartalma a következő: „Sarah is sleeping... Press the Start button to wake up.”.

```
1 class writerThread(QThread):
2     signal = pyqtSignal(str)
3     def __init__(self, parent = None):
4         QThread.__init__(self, parent)
5
6     def run(self):
7         global SarahSaid
8         global OldSarahSaid
9
10        global MasterSaid
11        global OldMasterSaid
12        while True:
13            if SarahSaid != OldSarahSaid:
14                try:
15                    self.signal.emit("Sarah: " + SarahSaid)
16                    SarahSaid = OldSarahSaid
17                except:
18                    logging.error('Same value')
19            if MasterSaid != OldMasterSaid:
20                try:
21                    self.signal.emit(MASTER + ": " + MasterSaid)
22                    MasterSaid = OldMasterSaid
23                except:
24                    logging.error('Same value')
```

3. Kódrészlet: A szövegdobozba írás

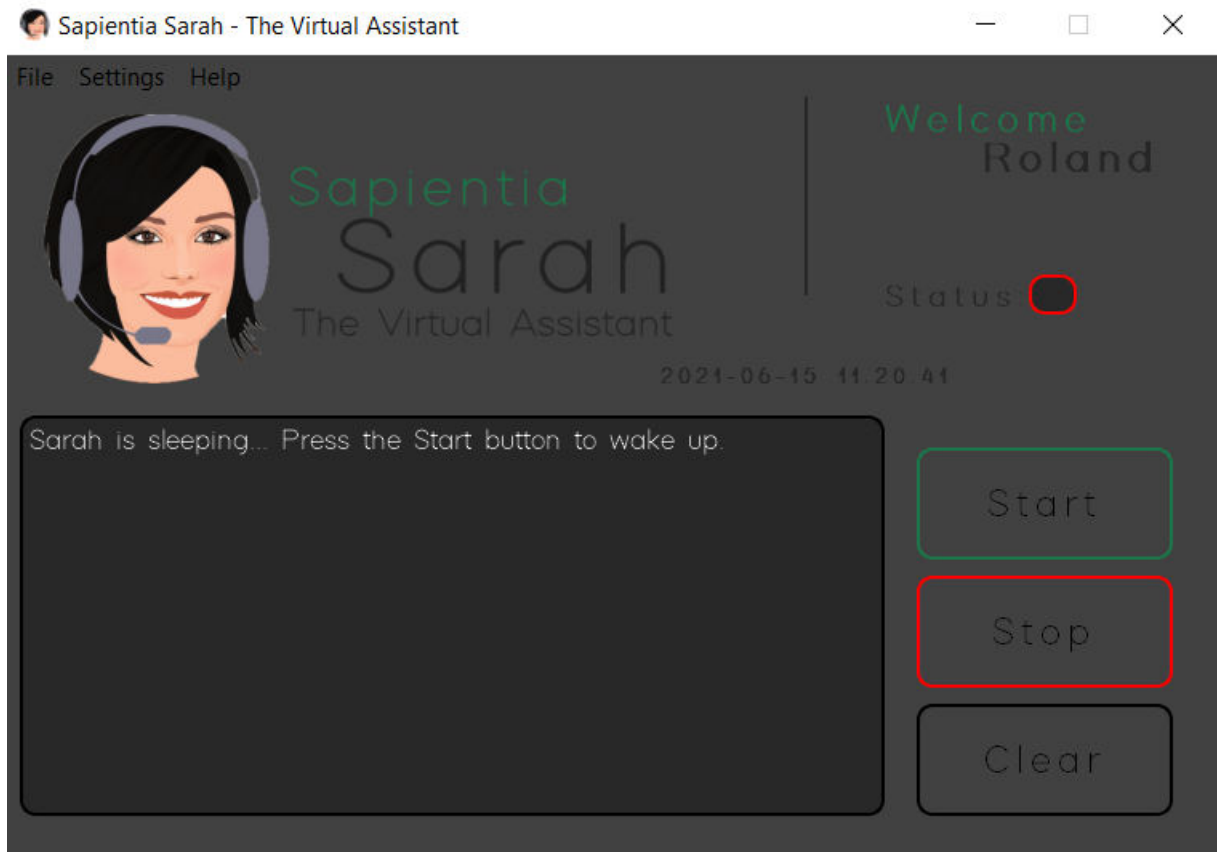
A 3-as kódrészlet esetén a *writerThread run()* függvényére kell figyelmet szentelni. Ez a függvény felel azért, hogy mindig az éppen aktuális és kellő szöveg kerüljön bele a szövegdobozba, amelyhez négy globális változóra volt szükség. A *SarahSaid* változó inicializálása abban a függvényben van megoldva, ahol a rendszer beszél, a lejátszott hangfájlokat megelőzően szöveges formátumban átadom a kellő információt a globális változónak. Az *OldSarahSaid* segédváltozó folyamatosan megkapja a már kiírt értéket biztonsági okokból, hogy kétszer ugyanaz az üzenet ne kerüljön ki a dobozba. A *MasterSaid* globális változó inicializálására a hangbemenet detektálásakor kerül sor. Amikor a rendszer a hangot feldolgozta és szöveges formátummá alakította, hogy a végrehajtás működni tudjon azonnal átadódik a globális változónak a string. Az *OldMasterSaid* szerepe teljesen megegyezik a *OldSarahSaid* szerepével. A szöveg beszúrás az *append* funkcióval történik.



11. Ábra: A komponens diagram¹¹

A továbbiakban sor kerül a UI elemek bemutatására, amelyek a 11. ábra első komponensében szerepelnek. Ezen elemek osztályonként funkcionálnak, ezek fellelhetők a 15 ábrán, a felhasználói felület alfejezetben.

Az összes felhasználói felületbéli elem külön ablakként funkcionál, egyszerre több előugró ablak is megjelenhet, ha a felhasználó elindítja. A fő QThread-en futó UI_SapientiaSarah alkomponens felel az összes többi létrehozásáért. Az egy komponens felépítése, már taglalva volt a felhasználói felület fejezetnél, ebben a részben a főbb ablakok gombjainak funkcionálisait, illetve az ablakok működését, kinézet, elosztását fogom bemutatni.



12. Ábra: A Main Screen

A Sapiaientia Sarah nevű alkalmazás főképernyője a `Ui_Sapiaientia_Sarah` komponens, több helyen Main Screen néven van rá hivatkozva a dolgozatban megtalálható a 12. ábrán. Amint elindul az alkalmazás ezzel az ablakkal találja magát szembe a felhasználó. Az ablak bal oldalán megtalálható az alkalmazás fejléce a logóval ellátva, a jobb oldalon egy üdvözlő üzenettel, név szerint, amelyet regisztrációnál adunk meg az alkalmazásnak, továbbá ezt módosíthatjuk is idő közben. A két oldalt egy úgynevezett design csík választja el egymástól. Az üdvözlő üzenet alatt található a státusz kocka, amelynek azon egyszer szerepe van, hogy zöldre vált a keret, amikor Sarah hallgat és beszélhetünk hozzá. Alapértelmezett állapotban, amíg a Start gomb nincs elindítva a státusz piros, amikor a rendszer beszél zöldre vált. Mindez a Use Case diagram Hang utasítást ad kiértékelő táblázatában ki lett fejtve.

Van egy dátum mező, amely másodpercre pontosan közli írásos formátumban a dátumot és a pontos időt a felhasználóval. Ezen egységet is egy `QThread` frissít folyamatosan, amely implementálása az 1. Kódrészletben található, az azt követő bekezdésben kerül sor a magyarázatára. A Stop gomb megállítja a hallgatást, a Clear törli a szöveges doboz tartalmát. Abban az esetben, amikor maga az asszisztensi rész elindításra kerül a Start gombbal, a szöveges doboz tartalmában és élesben is elkezdődik a konverzáció Sarah és a felhasználója között.

```
1 def start_button(self):
```

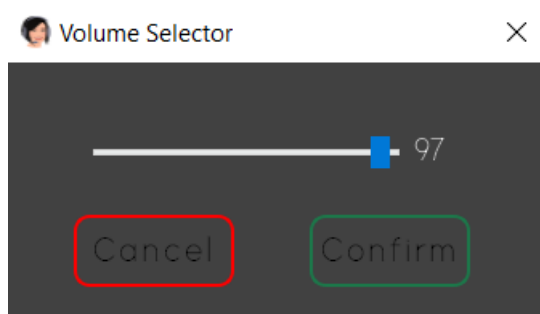
```

2      #starting thread
3      self.worker = workerThread()
4      self.worker.start()
5
6      #writer thread:
7      self.thread = writerThread()
8      self.thread.signal[str].connect(self.updateOutputTextUi)
9      self.thread.start()

```

4. Kódrészlet: A Main Screen Start gombja

A 4. kódrészletben lapul a Main Screen Start gombja. A gomb annyit csinál, hogy elsősorban elindítja 9. Kódrészletben fellelhető, majd az azt követő paragrafusban elmagyarázó *workerThread()*-et elindítja, továbbá Slot/Signal²⁰ mechanizmust használva elindítja a 3. Kódrészletben feltüntetett *writerThread()* osztály *run()* metódusát.



13. Ábra: A Volume Selector

A 13. ábrán a *Ui_Volume_Selector* komponenshez tartozó GUI található, amely működése rendkívül egyszerű. A csúszkát 0-tól 100-as skálán lehet mozgatni, abban az esetben, ha kiválasztottuk a hangerő értékét a „Confirm” gombra kell kattintanunk, hogy Sarah végrehajtsa nekünk a rendszer hangerejének újra kalibrálását. A „Cancel” gombbal a beállított értéktől függetlenül minden marad a régiben, az ablak bezáródik.

```

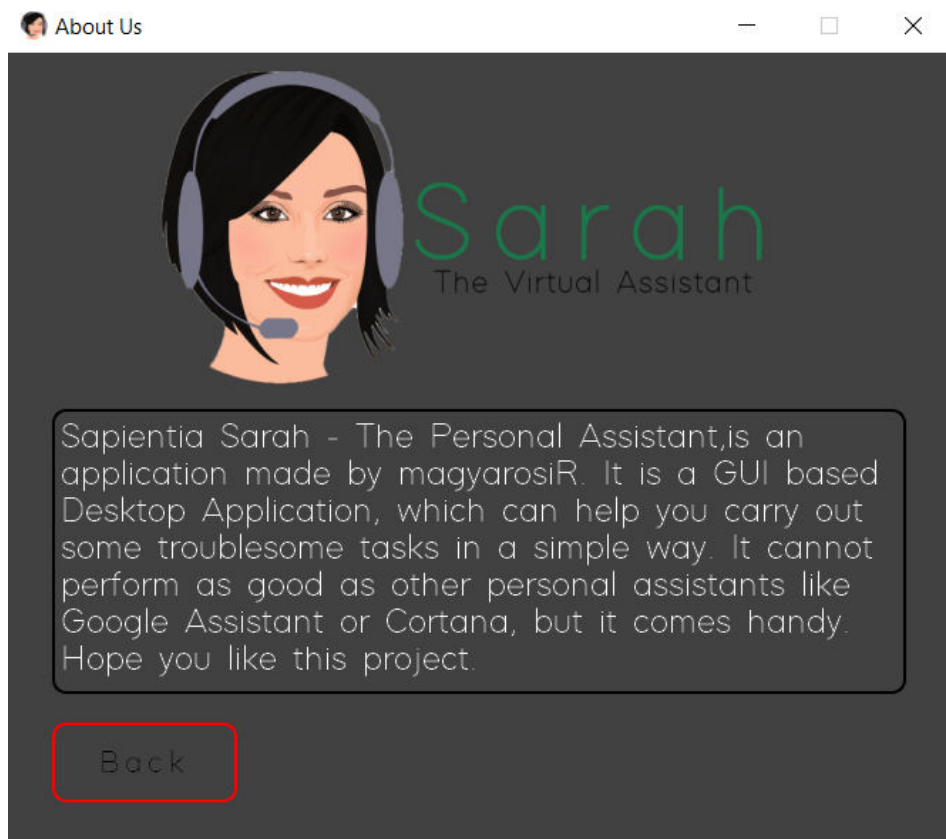
1      def cancel_button(self, x):
2          x.close()
3
4      def confirm_button(self, x):
5          value = self.VolumeSlider.value()
6          print (value)
7          keyboard = Controller()
8          for i in range(0, 50):
9              keyboard.press(Key.media_volume_down)
10             keyboard.release(Key.media_volume_down)
11
12             for i in range(0, int(value / 2)):
13                 keyboard.press(Key.media_volume_up)
14                 keyboard.release(Key.media_volume_up)
15             self.settings.setValue('volume', self.VolumeSlider.value())
16             x.close()

```

²⁰ Felhasználói felület alfejezet 13. paragrafusa

5. Kódrészlet: A Volume Selector gombjai

Az 5. kódrészlet szemlélteti a két függvényt, amely felel a gombok funkcionalitásaiért. A `cancel_button()` metódus bezárja az ablakot, míg a `confirm_button()` metódus a 18. Kódrészlethez hasonlóan a `keyboard könyvtár Controller` osztályát használja a hangerő módosításhoz gomblenyomás szimulációval. A `value` nevű lokális változóba lekérjük a csúszka értékét. Az első `for` ciklus azért felelős, hogy nullázza le a rendszer hangerejét, tehát a hanglehúzó gomb szimulálása addig tart, amíg a rendszer hangereje nullázódik, majd a hangfelhúzó gombbal beállítjuk a csúszkáról levett adatot és átadjuk a beállításoknak a `QSettings` osztálynak hála, amely állandó alkalmazásbeállítást biztosít.



14. Ábra: Az About Us ablak



15. Ábra: A Privacy Policy ablak

A 14., illetve a 15. ábrán az About Us, illetve Privacy Policy ablak látható, amelyek ugyanarra a sablonra épültek és a céljuk is megegyezik, mindkettő arra hivatott, hogy információt közöljön a felhasználóval, az előbbi magáról a Sapia Sarah alkalmazásról, míg az utóbbi az adatvédelmi feltételekről.

Ezen ablakok nem rendelkeznek különösebb funkcionalitással, a Back gomb mögött ugyanaz a kódsor áll mindkét esetben, mint a 5. Kódrészletben fellelhető programrész `cancel_button()` metódusában.



16. Ábra: A Send E-mail ablak

A 16. ábra szemlélteti azt az ablakot, amely az e-mail küldésért felelős. Az ablakban két bemeneti mező van, a felső azt az e-mail címet várja, akinek szeretnénk küldeni az üzenetet, az alsó pedig magát az üzenetet. Az e-mail mező le van kezelve, reguláris kifejezéssel, hogy a megadott érték formálisan megfeleljen a standard e-mail címeknek. A szöveges dobozba alapértelmezetten megjelenik a „Sent from Sapienia Sarah IPA” mondat, ezt közkívánatra ki lehet törölni.

Az ablak jobb oldalán három gomb található, név szerint a „Send”, a „Back”, illetve a „Clear”. A „Back” gombra nyomva az ablak eltűnik, vele az összes beírt adattal, a „Clear” törli az üzenet szöveges doboz tartalmát, a „Send” eléggé specifikus funkcióval bír, ugyanis itt történik az e-mail küldése, ez megtalálható az alábbi, 6. kódrészlet tartalmában.

```
1 try:
2     self.personal_details = QSettings(
3         'PyQt5Application', 'SapieniaSarah')
4
5     username = config['Personal']['email']
```

```

6     password = config['Personal']['password']
7
8     server = smtplib.SMTP('smtp.gmail.com', 587)
9     server.ehlo()
10    server.starttls()
11    server.login(username, password)
12    server.sendmail(username, email, message)
13    server.quit()
14    self.WarningLabel.setText("Your e-mail is sent!")
15    self.WarningLabel.setStyleSheet("color: rgb(28,120,74);")
16    self.WarningLabel.show()
17 except Exception as e:
18     self.WarningLabel.setText("I cant send your e-mail!")
19     self.WarningLabel.setStyleSheet("color: red;")
20     self.WarningLabel.show()

```

6. Kódrészlet: Az e-mail küldés²¹

Az e-mail küldés az SMTPLIB (Simple Mail Transfer Protocol Library) könyvtár segítségével történik. Az SMTP²¹ egy protokoll, amely segíti az e-mail küldést és fogadást a szerverek között, ezen aszimmetrikus protokoll a TCP/IP-n fut, azon belül is az applikációs rétegben. A függvényben reguláris kifejezéssel az e-mail forma ezúttal is le lett kezelve, továbbá az is ellenőrizve van, hogy egyik mező sem legyen üres küldéskor. Az email és a message lokális változókba le lett kérve a dobozok tartalma. ezt követően kezdetét vehette maga a küldés.

A config fájlból kiolvasom az SMTP által megkövetelt nyersanyagokat, amelyeket a regisztrációnál, illetve az adat módosításnál kell/lehet megadni.

A server változóba megadom az smtplib könyvtár SMTP osztályának a kapcsolathoz elvárt paramétereket, a hosztot karakterláncként, illetve a TLS (Transport Layer Security) portot. A TLS egy titkosítási protokoll, amelyek ahogyan az a nevéből is adódik biztonságot nyújt az interneten való kommunikációhoz.²¹

Az *elho()* metódus az azonosításért felel, pontosabban az e-mail szerver küld, amikor csatlakozik egy másik e-mail szerverhez önazonosításképp, hogy megkezdhesse az üzenet küldését. Az ELHO közli a fogadó szerverrel, hogy támogatja az ESTMP-vel kompatibilis kiterjesztéseket.

A könyvtár *starttls()* metódusának meghívásával a TLS²² titkosítás elkezdődik, tehát biztonságossá teszi az e-mailünket. A TLS protokollnak három fő összetevője van, a titkosítás, a hitelesítés, illetve az integrálás. A titkosítás egy harmadik fél elől elrejtí az információt, a hitelesítés biztosítja, hogy az információt cserélő felek azok legyenek, akiknek beállítják magukat, az integrálás ellenőrzi, hogy nem-e lettek manipulálva az adatok.

²¹ SMTP dokumentation: <https://docs.python.org/3/library/smtplib.html>

²² TLS: <https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/>

A *login()* metódus megkapja az inicializált paramétereket, a küldő e-mail címét, illetve jelszavát, a *sendemail()* a levél küldésért felel, az elvárt paraméterek, amelyeket meg kell adnunk, az a küldő e-mail címe a fogadó e-mail címe, illetve maga az üzenet. A *quit()* függvény befejezi az SMTP munkamenetet és lezárja a kapcsolatot.

Amennyiben a sorokon sikeresen végig ment Sarah egy label-ben zölddel tudatja a felhasználóval, hogy az általa küldött e-mail el lett küldve, amennyiben kivétel keletkezik vörössel az is említésre kerül, hogy képtelen elküldeni az üzenetet.



17. Ábra: Az adatok módosítása

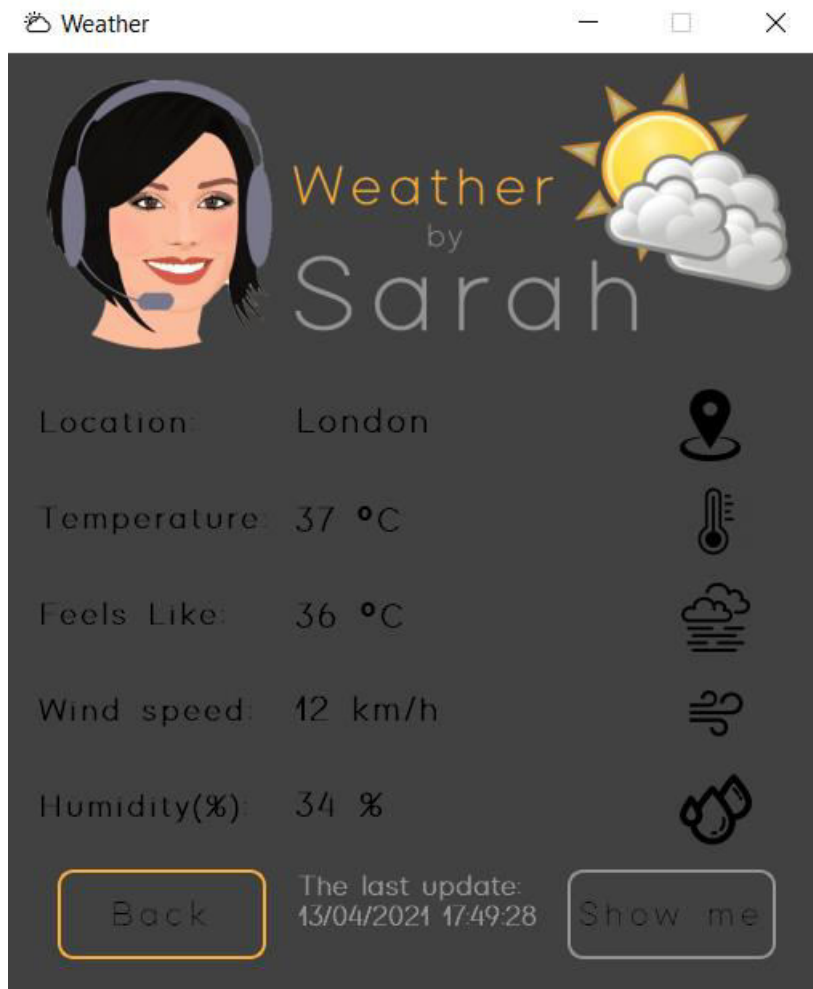
A 17. ábrán fellelhető az a Widget, amely a személyes adatok módosításáért felelős. Ebben a részben adjuk meg az adatainkat azért, hogy a teljes értékű rendszert használni tudjuk. A teljes nevünket, egy felhasználói nevet, a születési dátumot, a nemet, a valid e-mail címünket az

e-mail küldéshez, a jelszavunkat, illetve az átlagos lakhelyünket is azért, hogy navigálni tudjon bennünket Sarah. A lap végén el kell fogadnunk az adatvédelmi szabályzatot, ha sikeresen be akarjuk fejezni az adatmódosítást. A „Privacy Policy” felirat az gombként szolgál, tehát a Widgetet arra kattintva is meg tudjuk nyitani. A „Back” gombra kattintva nem kerülnek elmentésre az adatok, az az ablak bezárásáért felel, az „Ok” gomb viszont sokkal több ennél.

A gomb lenyomásával érvénybe lép az osztály `ok_button()` nevű metódusa, amely elsősorban lekéri az adatokat a szövegdobozokból, majd különféle szűrőknek kell megfelelni, ahhoz, hogy a config fájl tartalma frissülni tudjon. Ezen feltételek azok, hogy az összes cella ki legyen töltve, a születési év az ne haladja meg az aktuális dátumot, az e-mail cím formája valid legyen, valamint be legyen pipálva az adatvédelmi szabályzat esetében a checkbox. Amennyiben ezek nem teljesülnek egy úgynevezett „Warning label” pirossal jelzi a probléma okát. Amennyiben frissültek az adatok ezt zölddel tudatja a felhasználóval. Ha a feltételek teljesültek akkor frissül a config fájl tartalma, továbbá egyedül ebben az esetben generálódik új kulcs, tehát a mykey tartalma is megváltozik.



18. Ábra: Az időjárás widget aktiválás előtt



19. Ábra: Az időjárás widget aktiválás után

A 18-as, illetve a 19-es ábrán az időjárás ablak található meg, az elsőn a „Show me” gomb lenyomása előtt, míg a másodikon a gomb lenyomása után. A különbség abban nyilvánul meg, hogy a config fájlból lekérésre, illetve megjelenítésre kerülnek az adatok, illetve egy információs label frissül és leírja az utolsó lekérés időpontját, illetve dátumát.

4.3.A hangbemenet és beszédfeldolgozás

4.3.1. Áttekintés

A hangfelismerés első komponense természetesen a beszéd. A beszédet át kell alakítani szöveggé valamilyen módon. Először a mikrofon elektromos jellé alakítja azt, majd digitális adattá az analóg-digitál konverter segítségével. A digitalizálás után több modell is használható a szöveggé konvertáláshoz.

A legtöbb modern beszédfelismerésen alapuló modern rendszer az úgynevezett Hidden Markov Model-t (HMM) veszi alapul. Ez a megközelítés egy időbeli valószínűségi modell, amelyben a folyamat állapotát egy változó írja le. Ezt a modellt előszeretettel használják a hangfelismerés mellett többek között a fizikában, a kémiában, a jelfeldolgozásban, de a pénzügyekben és a mintafelismerésben is.

A beszéd szöveggé alakításához a vektorok csoportjait egy vagy több fonémához illesztik. A fonéma a nyelvi szintek közül a legalsó, önmagában a nyelvtudomány eszközeivel tovább nem bontható egység, amelyet fel lehet ismerni, ám érdemi jelentés nem társul hozzá. Ha csoportosítjuk, vagy összekapcsoljuk őket igencsak fontos szerep hárul rájuk. A fonémák felelnek azért, hogy meg tudjunk különböztetni egy-egy jelentést a másiktól. Ez a kalkuláció betanítást igényel, mivel a fonéma mikrofonról mikrofonra eltérő, sőt már fonémákat használnak a gép előtt ülő felhasználók is a hangjuk mássága miatt. A speciális algoritmust bevetve meghatározzuk a legvalószínűbb szavakat. [9]

A fent leírtak igencsak drága folyamat, ezért ezt olyan módon egyszerűsítették le, hogy a jelet, csak azokra a részekre redukálják, ahol nagy valószínűséggel beszéd történt, tehát a háttérzajt nem dolgozza fel a rendszer. Ezek segítségével a hangtevékenység-érzékelőt fejlesztették ki. Mindez akadályozza azt, hogy a rendszer időt pocsékoljon olyan felesleges részek elemzésével, amelyhez nem tartozik érdemi információ. [9]

4.3.2. A „SpeechRecognition” könyvtár

A SpeechRecognition python könyvtár rendkívül kedvelt a Python fejlesztők körében a csomag egyszerűsége miatt. Ezt a könyvtárat a mikrofon eléréshez készített különféle szkriptek helyett és a nulláról való feldolgozás helyett használtam. Ezen könyvtár több neves a témához kapcsolódó API alapjaként szolgál, ezek mellett rendkívül rugalmas. Ezen flexibilitásnak köszönhetően nagyszerűen illeszkedik bármiféle Python projektbe, többek között az enyémbe is. [9]

A Python 2.6-tól elérhető a könyvtár, amelyet a megszokott pip install parancssorral telepíthetünk is. Az általam használt csomag a legfrissebb 3.8.1-es verzió.

A telepítést és az importálást követően azonnal használhatóvá válik a könyvtár a szkriptben a *Recognizer()* osztály inicializálásával. Ezen osztály elsődleges célja természetesen a beszéd felismerése, ehhez létre kell hozni egy példányt, nagyon egyszerűen.

Minden esetben hét módszere van a beszédfelismeréshez, a projektben a *recognize_google()* API-t használtam erre a funkcióra, ugyanis ez az egyetlen megközelítés,

amihez nem kell hitelesítést alkalmazni, vagy bejelentkezéssel, vagy kulcslekéréssel. Egy másik megközelítés még a *recognize_sphinx()* lehetett volna, ugyanis annak az előnye az, hogy nincs szüksége internetkapcsolathoz a működéshez, azonban mivel a világhálózathoz való kapcsolat egy nagyon fontos nem-funkcionális követelmény így mérlegelve az előbb említett *recognize_google()* API optimálisabb volt. Az az egyetlen hátulütője, hogy napi 50 lekérésre van korlátozva, ám vélhetően ennél több indításra nincs szüksége egy felhasználónak naponta.

```
1 def takecmd():
2     with sr.Microphone() as source:
3         r.adjust_for_ambient_noise(source)
4         try:
5             while True:
6                 global statusBarColor, statusListening
7                 statusBarColor = False
8                 if statusListening == True:
9                     statusBarColor = True
10                    audio_queue.put(r.listen(source, phrase_time_limit=10))
11                    time.sleep(8)
12                else:
13                    pass
14            except KeyboardInterrupt:
15                stop_threads = True
16                pass
```

7. Kódrészlet: A hangbemenet elcsípése

A 7. kódrészletben fellelhető sorok felelnek a hallgatásért a backend főszálon. A könyvtár *Microphone()* metódusa hozzáfér a számítógép alapértelmezettnek állított mikrofonjához. A Recognizer osztály *adjust_for_ambient_noise* függvénye a zajkezelésért felel. Mivel a mikrofonból érkező bemenet általában jóval zajosabb, mintha egy előre előkészített audio fájlból próbálnánk értelmezni az adatokat célszerű kiszűrni a zajt.

Szemaforos módszerrel van blokkolva a folyamatos feldolgozott adat queue-be tétele annak érdekében, hogy ne történjen visszhangozás a rendszerben. Amikor *takecmd()* függvény életbe lép az blokkolja a főszálon futó hallgatást, hogy a rendszer válasza ne kerüljön bele a queue-be és ez ne kerüljön feldolgozásra. [10]

Amikor a szemafor úgy engedi a globálisan példányosított *audio_queue*-be bekerül a rögzített adat a könyvtár *listen()* függvényét használva. Ezen függvény egy *AudioData* objektumot helyez a várakozási sorba, amelyet a egy hátsó szálon dolgozunk fel a későbbiekben.

A Queue a veremhez hasonlóan egy lineáris adatstruktúra, egy tároló, egy várólista, amely az adatokat FIFO (First In First Out) módon tárolja. Ezen megközelítés feltételezi, hogy a várakozási listába először bekerült legrégebbi adat kerül ki először onnan feldolgozásra.

A *phrase_time_limit* a program várakozási idejét jelenti, pontosabban azt, hogy hány másodpercig vár a program az utasításra, mielőtt azt tovább küldené, ez jelen esetben 10 másodpercre van állítva.

A várakozási sor *put* metódusával szűrjük be az éppen aktuálisan értelmezett *AudioData* objektumot, amelyet a későbbiekben analizálunk.

```
1 def assistant():
2     while True:
3         audio = audio_queue.get()
4         global MasterSaid
5         if audio is None:
6             print('kileptek assistantban')
7             break
8         try:
9             incoming = r.recognize_google(audio)
10            MasterSaid = incoming
11
12            if 'time' in incoming or 'clock' in incoming:
13                TimeC.current_time()
14
15            elif 'date' in incoming:
16                TimeC.current_date()
17
18            elif 'open' in incoming:
19                InternetC.open_website(incoming)
20
```

8. Kódrészlet: A hangbemenet értelmezése

A 8. kódrészletben szemléltetve van az, hogy hogyan értelmezzük a várakozási listából kieszedett objektumot és azt szöveggé alakítva hogyan használjuk fel a parancsok végrehajtásában. A kódrészlet hosszúságára tekintettel mindössze egy része van feltüntetve.. Az *audio* változóba lekérjük a várakozási sor aktuális elemét a *get()* metódussal, amely alapértelmezetten lekérés után törli is a queue-ből. Itt jön képbe a már említett *recognize_google* API, amely értelmezi és szöveggé alakítja az objektumot. Ez az internetkapcsolatnak a sebességétől pár másodpercet vesz igénybe. A *recognize_google()* metódus mindig a legvalószínűbb szót küldi vissza, így előfordulhat, hogy rosszul értelmez valamit, sokszor érdemes a teljes választ megnézni, ehhez a *show_all* paramétert kell igazra állítani. Esetünkben nem volt erre szükség.

Az *incoming* változót már stringként tudjuk kezelni, így azt is tudjuk értelmezni, hogy helyet kapott-e az általunk kimondott szöveggé alakított mondatban egy-egy szó, ilyen például a 'time' vagy a 'date'. Amennyiben megjelent azonnal meg tudjuk hívni a nemes egyszerűséggel a példának okáért a *TimeC* osztály *current_time* statikus metódusát, amely jelen esetben azért felel, hogy elmondja az aktuális időt.

Ezen *assistant()* függvény egy hátsó szálon munkálkodik, amely akkor kerül elindításra, amikor a GUI főoldalán megnyomjuk a Start gombot, ekkor kel életre az egész rendszer.

```
1 class workerThread(QThread):
2     def run(self):
3         SystemC.wishme()
4         audio_thread = Thread(target=assistant)
5         audio_thread.daemon = True
6         audio_thread.start()
7         takecmd()
8         audio_queue.join()
9         audio_queue.put(None)
```

9. Kódrészlet: A feldolgozó szálat elindító QThread osztály

A 9. kódrészlet elénk tárja azt a módszert, ahogy elindításra kerül az *assistant* függvény a hátsó szálon. Ezt egy QThread munkásszámban kell elvégezni ahhoz, hogy ne zavarja a főszálat működés közben. A 17. Kódrészlet szemlélteti a *wishme()* statikus metódus meghívása arra szolgál, hogy amikor Sarah felébresztjük a Start gombbal, ő köszöntsön bennünket. Ezen metódus a rendszerrel kapcsolatos Brain modul 5. paragrafusában van elmagyarázva.

Az *audio_thread* változóban létrejön az új szál²³, paraméterként meg van adva a függvény, amelyet futtatni szeretnénk ezen a szálon. Esetünkben ez egy úgynevezett „daemon” szál²³ lesz, ugyanis nagyon fontos, hogy amikor a program futása megáll, a szál is azonnal álljon meg, nem kell aggódnunk és felelősséget vállalnunk a leállítása miatt. Ellenkező esetben a program addig vár, ameddig a szál be nem fejezi a munkálkodást, ám ez a végtelenig is eltarthat. Jelen esetben nagyon fontos az, hogy a hátsó szálnak véget kell szakadnia, amikor a rendszer megáll, vagy a felhasználó kilép a GUI-ból. Ezeket követően a *start()* metódussal elindítjuk a szálat, meghívjuk az előbb kivesézett *takecmd()* függvényt, valamint hozzacsatoljuk a Queue-t is a *join()*-nal.

4.3.3. Az asszisztens válasza, Sarah hangja

Az egyik legfőbb követelmény, amelynek meg kell felelnie a legtöbb virtuális asszisztensnek – és ez alól Sarah sem képez kivételt – az nem más, mint a kommunikáció. Egy asszisztensnek kell tudnia információt cserélni emberi nyelven, ez teszi őket emberibbé és nagy vonalakban személyi asszisztenssé. A kommunikáció hiánya szakszerűtlenné és effektív használhatatlanná teszi az alkalmazást. A gyenge kommunikációs képességekkel rendelkező asszisztensek is olyan hatást keltenek, mint azok, amelyek egyáltalán nem képesek kommunikálni.

²³ Multithreading, daemon threads:

https://www.bogotobogo.com/python/Multithread/python_multithreading_Daemon_join_method_threads.php

Esetünkben Sarah a Google által kifejlesztett GTTS-nek (Google-Text-to-Speech) hála beszélni is tud, továbbá a leírtakat a főmenüben (11. Ábra) megtalálható szövegdobozban írásos formátumban is jelzi a felhasználónak. Az asszisztens kommunikációs kétségei továbbfejleszthetők ezért választékosná vált Sarah szókincse. A GTTS könyvtár²⁴ rákapcsolódik a szintén Google által kifejlesztett Google fordító szöveget beszéddé alakító API-jához.

```
1      @staticmethod
2      def speak(string):
3          global SarahSaid, statusListening
4          tts = gTTS(text=string, lang='en')
5          r = random.randint(1, 20000000)
6          audio_file = 'audio' + str(r) + '.mp3'
7          tts.save(audio_file)
8          SarahSaid = string
9          print("Sarah Said: " + SarahSaid)
10         statusListening = False
11         playsound.playsound(audio_file)
12         statusListening = True
13         logging.info(f"Sarah: {audio_string}")
14         os.remove(audio_file)
15         logging.debug('Audio file removed.')
```

10. Kódrészlet: A beszédért felelős metódus

A System komponens *speak()* statikus metódusa felel rendszer-szerte a beszédért. Ennek a kivitelezése úgy történik, hogy a paraméterként megadott string értéket alakítom át Sarah hangjává és beszédévé. A tts változónak gTTS könyvtár főfüggvényét feleltetjük meg, az első paramétere maga a szöveg, a második a nyelv, esetünkben a függvényben, paraméterként megadott string, illetve az angol nyelv. A GTTS audio fájlra alakítja a szöveget, majd azt játssza le, ennek az elmentéséhez szükséges egy string, amely jelen esetben az *audio_file*, ez lesz a mentett fájl neve. A lejátszást követően az *os.remove()* törli a hangfájlokat, azonban amennyiben később a fejlesztő úgy dönt, hogy mégis meg szeretné tartani, akkor erre is lehetősége adódik, hiszen random generált számok kerülnek a fájlok nevébe egészen 1-től 20 millióig, így elenyésző annak a valószínűsége, hogy felülírás alá kerül bármi is. A *playsound* könyvtár²⁵ is itt jön képbe, amit lényegében arra használok, hogy lejátssza az elmentett hanganyag tartalmát.

A *SarahSaid*, illetve a *statusListening* globális változók ebben a függvényben váltanak értéket és nagyon fontos szerep hárul rájuk. Az előbbi azért felel, hogy folyamatosan jusson el a szöveg, amit Sarah kimond az kimeneti dobozba, ahol látható a konverzáció a felhasználó és maga Sarah között. A *statusListening* egy szemaforos megoldásként szolgál, ugyanis amikor

²⁴ gTTS: <https://pypi.org/project/gTTS/>

²⁵ Playsound: <https://pythonbasics.org/python-play-sound/>

Sarah beszél akkor nem kellene visszahallania magát egyidejűleg, így csak akkor hallgat, amikor visszállt a globális változó értéke igazra.

4.4. Brain modulok

Az agymodulok diagrammon szemléltetve megtalálhatók a 10. Ábra tartalmából, a komponens diagrammon a „Brain” részben. Az agymodulokat négy nagy részre osztottam, az idővel kapcsolatos komponensre, a „TimeC”-re, a leginkább az internetet igénybe vevő komponensre, az „InternetC”-re, a rendszerrel kapcsolatos modulra, a „SystemC”-re és végül az időjárásért felelős komponensre, a „WeatherComponent”-re. Ezen komponensek metódusai Sarah agyát képezik, ugyanis itt történik minden, ami végrehajtás! Ezen metódus hívások jól láthatók a 25. ábrán, illetve taglalva is vannak a „A „SpeechRecognition” könyvtár” fejezet 12. paragrafusában.

A modulokban fellelhető metódusok mindegyike statikus, ugyanis ezt a megközelítést láttam a legkifizetődöbnek úgy komplexitásban, mint idősporlásban, ugyanis egyszerűen meghívhatók és kezelhetők. A statikus metódusok ugyanis nem igényelnek példányosítást, tehát nem függenek az objektum aktuális állapotától. A statikus metódusok nem tudnak semmit az osztályról és csak a paraméterükre koncentrálnak.

A következőkben a 10. Ábrán fellelhető komponens diagram Brain moduljainak az almoduljait fogom bemutatni.

4.4.1. Idővel kapcsolatos

Az idővel kapcsolatos agymodul, amely osztálya a TimeC nevet kapta két metódust tudhat a magáénak. Egyet, ami az éppen aktuális időt téríti vissza, illetve egy másikat, amely a dátumért felel.

A teljes, viszonylag egyszerű és rövid osztály megtekinthető a lenti kódrészletben.

```
1 class TimeC(object):
2     @staticmethod
3     def current_time():
4         theTime = datetime.now().strftime("%H:%M")
5         SystemC.speak(f"The current Time is {theTime}")
6
7     @staticmethod
8     def current_date():
9         theDate = datetime.today().strftime("%B %d, %Y")
10        SystemC.speak(f"The current date is {theDate}")
11
```

11. Kódrészlet: A TimeC osztály teljes tartalma

A `current_time()` statikus metódus felel a pontos idő visszaadásáért. Egy lokális változóban adom meg a `datetime` könyvtár `now()` függvényét, amely visszatéríti az éppen aktuális, pontos időt, mindezt a `strftime()` függvénnyel formázzuk kedvünk szerint, jelen esetben az óra:perc a forma. A SystemC komponens `speak` metódusa elmondja Sarah nevében az aktuális időt. Ez a metódus akkor lép érvénybe, amikor a rendszer észleli a „time”, vagy a „clock” szót a bemenetben.

A `current_date()` statikus metódus azért felel, hogy közölje a felhasználójával az éppen aktuális dátumot, az előző funkcióhoz hasonlóan ez is a `datetime` könyvtárat használja, azonban ez a `today()` funkcióból nyeri a nyersanyagot. A közlés ugyanúgy a SystemC komponens beszédért felelős metódusával történik emberi nyelven. Akkor aktiválódik ezen függvény, amikor a „date” szócska megjelenik a hang bemenet részeként.

4.4.2. Internettel kapcsolatos

Az interneten való böngészésért felel ez az komponens, illetve olyan dolgokért, amelyeknek közvetlen köze van a böngészőhöz, valamint a világhálóhoz.

```
1 class InternetC(object):
2
3     @staticmethod
4     def open_website(string):
5         try :
6             website = string.split(' ')
7
8             webbrowser.open("https://" + website[website.index("open")+1] + ".com")
9             SystemC.speak(website[website.index("open")+1] + " is Opened ")
10        except Exception as e :
11            SystemC.speak("i can't see it")
12            logging.error('Unknown input')
```

12. Kódrészlet: Az InternetC osztály első metódusa

A 12. kódrészletben látható az InternetC osztály és annak első statikus metódusa, az `open_website()`, amely egy adott weboldal megnyitásáért felel. Az interneten való böngészés, weboldal megnyitás a leghasználtabb vonások közé tartozik ezért a mai virtuális asszisztensek esetében ez is alapkövetelménynek mondható. A weboldalak betöltését Sarah esetében a „webbrowser” könyvtárral²⁶ oldottam meg.

Ezen metódus akkor kerül meghívásra, amikor az „open” szócskát ejti ki a felhasználó. Amint feldolgozásra került a bemenet egy lokális változóban feldaraboljuk azt szóközönként és egy listába helyezzük. A weboldal formázását paraméterként adjuk meg, olyan módon, hogy a

²⁶ Webbrowser: <https://docs.python.org/3/library/webbrowser.html>

lista „open” szócska utáni elemét teszi a „https” illetve a „com” domain közé. A webbrower könyvtárnak hála egyszerűen az *open()* függvényével meg tudjuk nyitni a websiteot. A beszéd metódusnak köszönhetően Sarah ugyanazon listában feldarabolás végett ismeri a weboldalt, így el tudja mondani, hogy melyik oldal lett megnyitva. Amennyiben probléma adódik hibaüzenetet közöl velünk.

```
1  @staticmethod
2  def price_of_big_companies(string):
3      search_term = string.lower().split(" of ")[-1].strip()
4      stocks = {
5          "apple": "AAPL",
6          "microsoft": "MSFT",
7          "facebook": "FB",
8          "tesla": "TSLA",
9          "amazon": "AMZN",
10         "facebook": "FB",
11         "walmart": "WMT",
12         "ford": "F",
13         "intel": "INTC",
14         "oracle": "ORCL",
15         "nike": "NKE",
16         "adidas": "ADDYY",
17         "coca-cola": "COKE",
18         "netflix": "NFLX",
19         "starbucks": "SBUX",
20         "bitcoin": "BTC-USD"
21     }
22     try:
23         stock = stocks[search_term]
24         stock = yf.Ticker(stock)
25         price = stock.info["regularMarketPrice"]
26         SystemC.speak(f'price of {search_term} is {price}')
27     except:
28         SystemC.speak('oops, something went wrong')
```

13. Kódrészlet: A nagy vállalatok értékének megkeresése

Sokak számára fontos funkció az, hogy gyorsan elérjék a világ legnagyobb márkáinak, termékeinek aktuális értékét, összegyűjtöttem 2021 legnagyobb és legismertebb cégeit, az *yfinance*²⁷ könyvtárnak hála meg egy API-n keresztül le tudom kérni azoknak az aktuális értékét. A Yahoo által fejlesztett nyílt forráskódú könyvtárban rengeteg adat és statisztika fellelhető a nagy cégekről, viszonylag gyorsan, a perc töredéke alatt elérjük az API-t és választ is kapunk. Internetsebességtől függően megközelítőleg másfél másodpercbe kerül a legkérés, választól. A 13. kódrészletben fellelhető metódus első sorában az eddigiekhez hasonlóan kiszűrjük a szóközöket és megfeleltetjük neki a beérkező szöveg „of” szócska utáni szöveget, mindezt

²⁷ Yahoo finance: <https://analyticsindiamag.com/hands-on-guide-to-using-yfinance-api-in-python/>

kisbetűvé alakítva. Ezt követően létrehoztam egy kulcs-értékpárokkal felépített listát, a jobb oldalon az észlelhető bemenetek vannak, a jobb oldalán az API-ból lekért szimbólumok, amelyek szerint keresünk. Amint ez megvan és a listában megkapjuk a keresett vállalat nevét a `yfinance Ticker()` metódusát felhasználva lekérjük a szimbólum alapján az adatokat, ezt objektumként kapjuk meg, amely egy nagy adathalmaz, az összes adattal, ami a felhőben van tárolva, majd a `price` lokális változóban hivatkozunk a „regularMarketPrice”-ra, ami visszaadja az éppen aktuális értéket. Ezeket követően már csak annyi maradt hátra, hogy mindezt közvetítsük a felhasználó felé, illetve lekezeljük azt az esetet, amikor hiba történik.

```

1  @staticmethod
2  def search_google_youtube_wiki(string):
3      try:
4          splitted = string.split(' ')
5          theKey = string.lower().split(" about ")[-1].strip()
6          if "Google" in splitted or "google" in splitted:
7              webbrowser.open("https://google.com/search?q="+theKey)
8              SystemC.speak(f"Here is what I got about {theKey}.")
9
10         if "youtube" in splitted or "YouTube" in splitted or "Youtube"
11 in splitted:
12             webbrowser.open("https://youtube.com/search?q="+theKey)
13             SystemC.speak(f"Here is what I got about {theKey}.")
14
15         if "Wikipedia" in splitted or "Wiki" in splitted or "wikipedia"
16 in splitted:
17             webbrowser.open("https://en.wikipedia.org/wiki/"+theKey)
18             SystemC.speak(f"Here is what I got about {theKey}.")
19     except:
20         SystemC.speak("Please try again.")

```

14. Kódrészlet: Keresés az interneten

A 14. kódrészlet szemlélteti Sarah azon modulját, amely az interneten való keresésért felel. Attól függően, hogy a felhasználója hogyan adja a parancsot neki, Ő képes a Google-n, a Youtube-n, illetve a Wikipédián is keresni, a találatokat pedig élénk tárja a kijelzőre. A hangbemenet kezelése a már megszokottnak mondható szavankénti listába helyezéssel kezdődik, az „about” szócska után elmondott úgynevezett kulcsra keres rá Sarah annak függvényében, hogy melyik nagy keresőgépezet nevét mondjuk a mondatukban. A megnyitás ebben az esetben is az Internettel kapcsolatos agymodul 3. paragrafusában kivesézett webbrowser könyvtárral történik. A `search_google_youtube_wiki()` funkciót a „search” szócska kelti életre.

```

1  @staticmethod
2  def search_location_in_google_maps(string):
3      if 'from' in string:
4          try:
5              splitted = string.split(' ')
6              fromLoc = splitted[splitted.index("from")+1]
7              toLoc = splitted[splitted.index("to")+1]

```



```

8
9         webbrowser.open
10 (f'https://www.google.com/maps/dir/{fromLoc}/{toLoc}')
11         SystemC.speak("Here is the most optimal route")
12     except:
13         SystemC.speak("I couldn't find the place")
14 else:
15     try:
16         splitted = string.split(' ')
17         loaded_key = Crypto.key_loader('mykey.key')
18         Crypto.decryptor(loaded_key, 'config.ini')
19         fromLoc = config['Personal']['location']
20         toLoc = splitted[splitted.index("to")+1]
21
22         webbrowser.open
23 (f'https://www.google.com/maps/dir/{fromLoc}/{toLoc}')
24         SystemC.speak("Here is the most optimal route")
25         Crypto.encryptor(loaded_key, 'config.ini')
26     except:
27         SystemC.speak("I couldn't find the place")

```

15. Kódrészlet: Navigálás a Google Maps²⁸ segítségével

Sarah egyik legfőbb kvalitása az az, hogy el tud bennünket navigálni egyik pontról egy másikhoz, ehhez a Google térkép weboldalas változatát²⁸ véve alapul, ennek a modulnak a forráskódja fellelhető a 15. kódrészletben. Ezen módozat a „move”, illetve a „navigate” opciókkal válik élessé, továbbá két lehetőséget tár elénk. Amennyiben a „from” szócska szerepel a bemeneti szövegben az azt követő helyszínről navigál minket a „to” szó utáni helyszínre. Mindez a listás módszerrel valósul meg, továbbá az Internettel kapcsolatos agymodul 3. paragrafusában már taglalt webbrowser könyvtárral.

Ellenkező esetben, amikor csak a „to” szó szerepel a Sarah által értelmezett mondatban az, ha elágazás különben ágára lépünk és ott az előre definiált úgymond átlagos lokációról röpít minket egy másik helyszínre, amelyet megadunk a „to” után. Az előbb említett átlagos lokációt a regisztráció során tudjuk megadni a rendszernek, ezt később természetesen módosíthatjuk is a „Personal” komponensből. A személyes adatokból való lokációs nyersanyag kiszűrése a Kriptográfia fejezet Fernet módszer alfejezetének utolsó, azaz 9. paragrafusában van taglalva.

Mindkét esetben le van kezelve a hibalehetőség, amennyiben nem tudja Sarah valamilyen oknál kifolyóan végrehajtani a parancsot a felhasználó tudtára adja és az ellenkezőjét is.

Az metódus akkor kerül aktiválásra, amikor a felhasználó kimondja a „weather” szót, ezt követően az „in” szó utáni lokáció kerül elmentésre, valamint keresésre azt feltételezve, hogy az a kívánt lokáció.

²⁸ Google maps: <https://www.google.com/maps>

4.4.3. Rendszerrel kapcsolatos

Ebben a komponensben az olyan metódusok vannak csoportosítva, amelyek a rendszerrel kapcsolatosak. Ezen komponens osztály az összes közül a legtöbb metódussal rendelkezik szám szerint 5-tel, valamint messze a legterjedelmesebb.

```
1 class SystemC(object):
2
3     @staticmethod
4     def sys_out(string=False):
5         SystemC.speak(f"My pleasure to help you {MASTER}, See you later")
6         logging.debug("The program is exited.")
7         os._exit(1)
```

16. Kódrészlet: Kilépés a rendszerből

A 16. kódrészlet szemlélteti az osztály objektumot, illetve az első statikus metódusát, amely a `sys_out()` nevet kapta. Ezen funkció abban az esetben aktiválódik, ha a rendszer észleli, hogy búcsúzni akarunk tőle, tehát olyan szavakat használunk, amelyek erre utalnak. Ezek mellett a program egyéb részeiben is használtam a metódust, amikor gombbal való kilépést implementáltam.

Jelen esetben nincs szükségünk a bemeneti, értelmezett hanganyagra, a 16. kódrészlet fellelhető metódus meghívásával elkészön Sarah a felhasználójától az előre definiált felhasználónevet igénybe véve. Az `os` könyvtár `_exit()` metódusát felhasználva kilépünk az 1-es megadott státusszal. A státusz egy egész típusú érték lehet, ez a státusz meghatározza a kilépés módját.

A 4.3.3-as alfejezet alatt taglalt, 10. Kódrészletben fellelhető `speak()` metódus is ezen modulhoz tartozik.

```
1     @staticmethod
2     def wishme(string=False):
3         hour = int(datetime.now().hour)
4         if hour >= 0 and hour < 12:
5             SystemC.speak("Good Morning " + MASTER)
6         elif hour >= 12 and hour < 18:
7             SystemC.speak("Good Afternoon " + MASTER)
8         else:
9             SystemC.speak("Good Evening " + MASTER)
```

17. Kódrészlet: A köszöntés

A 17. kódrészletben a `wishme()` statikus metódus felel, ami a „SpecchRecognition” könyvtár alfejezet 14. paragrafusában kerül meghívásra, a rendszer indításakor. A szerepe az, hogy időtől függően Sarah hogyan köszöntse a felhasználót, aki éppen életre keltette. Egy lokális változóba elmentésre került az éppen aktuális idő, egész számmá alakítva. Az időpont lekéréséhez a `datetime` könyvtár `now()` függvényét használtam, ez mellett a `hour` osztályból

kértem le a pontos órát. Ezeket követően lebontottam 3 nagy részre időtől függően, reggelre, délutánra, illetve estére. A reggel az éjfélről délelőttig tart, a délután déltől 18 óráig, ettől éjfélig estét köszön Sarah a felhasználónak, az előre definiált felhasználónévvel.

```
1 @staticmethod
2 def mute_unmute_sys(string=False):
3     global MASTER
4     keyboard = Controller()
5     keyboard.press(Key.media_volume_mute)
6     keyboard.release(Key.media_volume_mute)
7     SystemC.speak(f"OK {MASTER}")
```

18. Kódrészlet: A rendszer lenémítése

A 18. kódrészletben az a függvény szerepel, amely a rendszer lenémítéséért felel. Akkor aktiválódik, amikor a „mute”, vagy az „unmute” szót használja a felhasználó.

A művelethez a Python keyboard könyvtárának²⁹ Controller osztályát használtam, amely lényegében billentyű lenyomást szimulál. A *press* metódus lenyomja a paraméterként megadott gombot, a *release* meg elengedi. A billentyű `media_volume_mute` gombját használtam, amely a rendszer elnémításáról, illetve a feloldásáról kezkesedik.

```
@staticmethod
def start_application(string):
1     print(string)
2     try:
3         if 'chrome' in string or 'web browser' in string or 'browser'
4 in string or 'internet' in string or 'Chrome' in string:
5             try:
6                 os.startfile('chrome.exe')
7                 SystemC.speak("I'm started Chrome!")
8             except:
9                 SystemC.speak("Error! I'm started your default
10 browser!")
11                 os.startfile('iexplore.exe')
12
13
14         elif 'opera' in string or 'Opera' in string:
15             try:
16                 os.startfile('opera.exe')
17                 SystemC.speak("I'm started Opera!")
18             except:
19                 SystemC.speak("Error! I'm started your default
20 browser!")
21                 os.startfile('iexplore.exe')
22
23         elif 'internet explore' in string or 'edge' in string or
24 'Explore' in string:
25             os.startfile('iexplore.exe')
26             SystemC.speak("I'm started your browser!")
```

19. Kódrészlet: Az alkalmazások megnyitása

²⁹ Keyboard library: <https://pypi.org/project/keyboard/>

A 19. kódrészlet tartalma a *start_application* függvény egy részlete, amely azért felel, hogy applikációkat nyisson meg. Ezen függvény akkor aktiválódik, amikor a felhasználó szájából elhangzik a „start”, „run”, vagy „launch” szó. Ezt követően a bemeneti szövegnek kell tartalmaznia azt a programnevet, amit éppen meg akar nyitani. A példán jelen esetben többek között a „chrome” megnyitására kerül sor.

Ezen rész azért felel, hogy a számítógép alapértelmezett, leghasználtabb és legkedveltebb applikációit meg tudja nyitni Sarah parancsszóra. A kérdőív készítése közben nagy hangsúlyt fektettem erre a részre, tudni akartam, hogy az átlagfelhasználó milyen programokat használ a számítógépén. A kiértékelést követően nem érkezett meglepetés, Messenger, Chrome és társai voltak a leggyakoribb válaszok. Sarah jelenleg 26 alkalmazás megnyitására képes a komponens ezen függvényé nek hála.

A programok megnyitása az os könyvtár startfile metódusának, illetve az annál ritkábban, pontosabban mindössze kétszer használt system osztály.

A startfile azért felel, hogy megnyisson olyan programokat, amelyek a számítógépre vannak telepítve, mindezt az indítófájljuk nevének megadásával, illetve a kiterjesztésével, mindezt paraméterként kell átadnunk a függvénynek. A Chrome megnyitása esetében nem alapértelmezett program a számítógépen, így amennyiben az nincs telepítve, Sarah szól, hogy probléma adódott és új alternatíva után néz, ezen alternatíva az alapértelmezett Windows böngészőnek, az Explore-nek a megnyitása.

Az os könyvtár system metódusát két esetben használtam alternatívaként, amikor nem működhetett a startfile. Az os könyvtár system funkciója az operációs rendszerrel kapcsolatos parancsokat hajtja végre, a függvény paraméterként, karakterláncként adtuk meg, egy végrehajtó kódot, ha úgy tetszik.

A beállítások megnyitása esetében a 'start ms-settings:' karakterláncot kapta paraméterként a függvény. Ilyen módon nyithatunk meg kódból alapértelmezett Microsoft termékeket, legyen az a beállítások ablak, vagy bármi más, ami ehhez a kategóriához tartozik. Amennyiben valamilyen oknál kifolyóan nem tudja a parancsunkat értelmezni, vagy végrehajtani, kivitelezni Sarah, a beállítások helyett alternatívát keres, olyant, mint a Control Panel.

A saját gép megnyitásakor egyenesen a parancs ikonra koncentráltam, a kód, amely a paraméterként meg van adva a „This PC” parancsinkonját jelöli, így a system függvény könnyedén meg tudja nyitani azt.

4.4.4. Időjárással kapcsolatos

Ezen komponens az időjárás lekéréésért felel. Az általam elkészített kérdőív kiértékelése során megannyi visszajelzés érkezett arról, hogy a felhasználók szeretnék, ha Sarah tudná közölni számukra az aktuális időjárás, ez volt a második legnépszerűbb funkció (52.3%) az alkalmazások megnyitását követően (59.1%).

Az időjárást az openweathermap API közvetíti a felhasználók felé, pontosabban ezen API-t használva tudja Sarah az aktuális időjárást. Az OpenWeather³⁰ vállalat tulajdonában álló online szolgáltatás percről percre frissíti az adatait, majd közli egy API kulcson keresztül. Az előrejelzések konvolúciós gépi tanulást alkalmazva jönnek létre, különféle nyersanyagokat figyelembe véve, például meteorológiai állomások, reptéri meteorológiai állomások, földi radar állomások, illetve különféle műholdak.

```
class WeatherComponent(object):

    @staticmethod
    def current_weather(string):
        try:
            user_api = "decda835a08a732bf560fba09dd3b86d"
            splittedstring = string.split(' ')
            location = splittedstring[splittedstring.index("in")+1]
            complete_api_link =
"https://api.openweathermap.org/data/2.5/weather?q="+location+"&appid="+user_
api

            api_link = requests.get(complete_api_link)
            api_data = api_link.json()

            #create variables to store and display data
            temp_city = round(((api_data['main']['temp']) - 273), 3)
            hmdt = round(api_data['main']['humidity'], 3)
            wind_spd = api_data['wind']['speed']
            feels_like = round(((api_data['main']['feels_like']) - 273), 3)
            now = datetime.now()
            dateString = now.strftime("%d/%m/%Y %H:%M:%S")

            loaded_key=Crypto.key_loader('mykey.key')
            Crypto.decryptor(loaded_key, 'config.ini')
            #config.add_section('Weather')
            config.set('Weather', 'location', location)
            config.set('Weather', 'tempreture-in-city', temp_city)
            config.set('Weather', 'humidity', hmdt)
            config.set('Weather', 'feels-like', feels_like)
            config.set('Weather', 'wind-speed', wind_spd)
            config.set('Weather', 'date', dateString)
            with open(file, 'w') as configfile:
                config.write(configfile)
            Crypto.encryptor(loaded_key, 'config.ini')

            SystemC.speak("Please check the Weather GUI.")
```

³⁰ OpenWeatherMap: <https://openweathermap.org/>

```
except Exception as e:  
    SystemC.speak("Error")
```

20. Kódrészlet: Időjárás lekérés a felhőből

A 20. kódrészlet kódrészletben fellelhető a komponensért felelő osztály, illetve annak az egyetlen metódusa, a *current_weather()*. A hangbemenet részekre lett osztva, illetve listába helyezbe szóközönként, majd a location változó ba beletesszük az “in” szócska utáni szót, a lokációt, amelyet később felhasználunk a link generálásnál, amelyhez az API kulcs is szükség van.

A lekéréshez a requests könyvtár³¹ *get()* metódusát használtam, amellyel GET kérést küldtem az előre összerakott URL-re. A GET szerepe nem más, minthogy adatokat kérjen egy meghatározott erőforrástól, jelen esetben az előre összeállított linkből. A json³² könyvtár *.json()* metódusának hála listaként kezelhettem a visszatérített adatokat, így a lokális változókba könnyedén elmenthettem a nyersanyagokat, amelyeket később belepakoltam a config fájlba. Amennyiben sikeres volt a végrehajtás Sarah közli velünk, és megkér angol nyelven, hogy nézzük meg az ezért felelős 17. Ábrán fellelhető GUI-t, ahol megnézhetjük az elénk tárt adatokat. Egy dátum string-be elmentem a lekérés pontos dátumát és idejét, hogy a felhasználó azzal is tisztában legyen.

Ahhoz, hogy a nálunk standardnak mondható Celsius fokban tudja közölni Sarah az időjárási adatokat, ilyen például a hőérzet, illetve a tényleges hőmérséklet át kell alakítanom Kelvin-ből, azaz a lekért adatból ki kell vonnom a különbséget, a 273-at, a fájlba már ez az eredmény kerül be, mindezt 3 tizedesnyi pontosságra kerekítve.

4.5. Kriptográfia

4.5.1. Áttekintés

Az elmúlt években a kriptográfia, mint tudomány önállóvá vált és robbanásszerű fejlődésen ment keresztül. Mindez többek között annak tudható be, hogy az információs technológiák az évek során egyre inkább betörték a gazdaságba. Az információ érték és gyakran érzékeny olyan tekintetben, hogy ha illetéktelen kezekbe kerül anyagi, vagy erkölcsi károkat hagyhat maga után, a biztonsági résekről nem is beszélve. Az információ biztonságát

³¹ Python request: <https://realpython.com/python-requests/>

³² Json library: https://www.w3schools.com/python/python_json.asp

algoritmikus, rendszabályi és végül, de nem utolsó sorban fizikai technikák összefésülésével érjük el, a kriptográfia az algoritmikus biztonsági módszerek tudománya. [13]

A kulcskezelés vitathatatlanul a kriptográfia egyik legfontosabb része. Egy rosszul kiválasztott, vagy kigenerált kulcs megkönnyítheti az esetleges támadó dolgát. A kulcsgenerálás általában véletlenszerűen történik automatikusan, azt feltételezve, hogy ez kellően erős, a kulcsgenerálás biztonságos formája lehet, ellenkező esetben könnyen veszélybe kerülhet az információ.

4.5.2. A Fernet módszer

A Fernet³³ módszer a Python *cryptography* könyvtárának egy része, így elengedhetetlen a csomag telepítése.

Ez a módszer szimmetrikus titkosításon/visszafejtésen alapszik. Ez a megközelítés az információk rejtjelezésének legősibb módja, továbbá ez a legegyszerűbb is, a lényege az, hogy ugyanazzal a kulccsal történik a visszafejtés, amellyel az adott információ rejtjelezve volt.

A Fernet könyvtárra esett a választásom, ugyanis egyszerű a használata, illetve biztonságos is, mivel garantálni tudja, hogy a használatával a titkosított üzenetet nem lehet manipulálni, illetve visszafejteni a kellő kulcs nélkül.

A titkosításhoz tehát szükségünk van egy titkos kulcsra, amely jelent esetben véletlenszerűen generálódik, ezen bájtokból álló kulcs szükséges az adat kriptálásához és visszafejtéséhez is egyaránt. A kulcsot többször fel tudjuk használni, Sarah esetében több indításkor is ugyanazon kulcs felel a titkosításért, illetve a visszafejtésért egyaránt, ugyanis csak akkor generálódik új kulcs, amikor frissítjük a személyes adatainkat, csak ebben az esetben frissül a „myKey.key” fájl tartalma. A kulcs azért került lementésre, mert korábban olyan problémával találtam szemben magam, hogy a szimmetrikus kulcskezelés miatt a program újraindítását követően új kulcs generálódott, amellyel nem tudta visszafejteni a program a számára érdekelt adatokat. A kulcs elmentése egy .key fájlba történik, amely egy biztonságos módja a kulcstárolásnak.

```
1  class Crypto():
2      @staticmethod
3      def key_generator():
4          key = Fernet.generate_key()
5          return key
6
7      @staticmethod
8      def key_writer(key, where):
```

³³ Fernet: <https://cryptography.io/en/latest/fernet/>

```

9         with open(where, 'wb') as myKey:
10             myKey.write(key)
11
12     @staticmethod
13     def key_loader(key_name):
14         with open(key_name, 'rb') as myKey:
15             key = myKey.read()
16             return key
17
18     @staticmethod
19     def encryptor(key, original_file):
20         f = Fernet(key)
21
22         with open(original_file, 'rb') as file:
23             original = file.read()
24
25         encrypted = f.encrypt(original)
26
27         with open(original_file, 'wb') as file:
28             file.write(encrypted)
29
30     @staticmethod
31     def decryptor(key, encrypted_file):
32         f = Fernet(key)
33
34         with open(encrypted_file, 'rb') as file:
35             encrypted = file.read()
36
37         decrypted = f.decrypt(encrypted)
38
39         with open(encrypted_file, 'wb') as file:
40             file.write(decrypted)

```

21. Kódrészlet: A Crypto komponens osztálya³⁴

Sarah esetében a 21. kódrészlet szemlélteti és egyben lefedi a program kriptográfiai részét, statikus metódusokkal könnyedén meghívhatók az osztályfüggvények bárhol, ahol éppen enkriptálás, vagy dekriptálásra van szükség. Ezen részlet megírásához ihletet merítettem egy dokumentációból³⁴. A *key_generator()* függvény a kulcsgenerálásért felel, ehhez a *generate_key()* metódusát hívjuk meg, valamint adjuk át egy lokális változónak, amelyet vissza is térítünk. az importálást követően.

A *key_writer()*, illetve a *key_loader()* hasonlóképpen működik, annyi különbséggel, hogy amíg az első beleírja a kulcsot a paraméterként megadott fájlba, addig a második kiolvassa azt onnan.

Az *encryptor()* függvényem titkosít, a *decryptor()* visszafejt. Mindkét esetben kiolvassuk a fájl tartalmát, azt követően a titkosított/visszafejtett szöveget helyezzük bele. A Fernet beépített függvényei, az *encrypt*, illetve a *decrypt* segít ezen problémák megoldásához

³⁴ Object Oriented Fernet: <https://python-bloggers.com/2020/09/encrypt-and-decrypt-files-using-python/>

viszonylag egyszerűen és gyorsan. Ezen függvények a paraméterül megkapott adattal dolgoznak és rendkívül erős adatvédelmi és hitelességi garanciát vállalnak.

A Fernet 128 bites AES (Advances Encryption Standard) algoritmust használ a titkosításhoz CBC (Cipher Block Chaining) módban, SHA256 (Secure Hash Algorithm 25) hasító függvénnyel. A kriptográfiával foglalkozó fejlesztők körében nagyon elterjedt az AES algoritmus használata, ugyanis ez volt a kétezres évek elején az a titkosítási módszer, amely maga mögé utasította a DES-t, illetve 3DES-t a kriptográfia éléről először az USA-ban, amelyeket egyébként sikerült feltörni az összes kulcs kipróbálásának módszerével. Az AES esetében a blokkméret 128 bit, a kulcs azonban eltérő lehet, 128, 192, vagy 256 bit. Az AES iteratív blokkrejtjelező, amelynek rétegbe szerveződése követi a shanoni helyettesítéses-permutáció elvet. [11] [12]

```
1 loaded_key=Crypto.key_loader('mykey.key')
2 Crypto.decryptor(loaded_key, 'config.ini')
3 file = 'config.ini'
4 config = ConfigParser()
5 config.read(file)
6 MASTER = config['Personal']['nickname']
7 Crypto.encryptor(loaded_key, 'config.ini')
```

22. Kódrészlet: A nyers anyag kinyerése a config fájlból

Az 22. kódrészletben található példa szemlélteti azt, hogy hogyan lett kifejtve az egyébként titkosított config fájl tartalma azokon a részekén, ahol a rendszernek szüksége volt a személyes adatokra. A loaded_key változóba lekértem a mykey.key tartalmát, ehhez a 21. Kódrészletben fellelhető Crypto osztály key_loader() statikus metódusát alkalmaztam, amelynek működése a Kriptográfia fejezet Fernet módszer 6. paragrafusában van taglalva. Ugyanezen fejezet 7. bekezdésében szó esik az decryptor, illetve az encryptor függvényekről. Az előbbi meghívásával a adatainak tartalma elérhető lesz arra az időre, amíg jelen esetben a MASTER, azaz a felhasználó regisztrációnál előre definiált nevét megkapja. Ezt követően az encryptor meghívásával a fájl tartalma ismét kriptálva lesz.

5. Üzembe helyezés

5.1.Felmerülő problémák és megoldásaik

A szoftver tervezése közben felmerülő problémák és azok megoldásaik leírására kerül sor a következőkben.

Az egyik legnagyobb problémát a rendszer visszhangja okozta, amelyet az egy szálon való hallgatás, illetve egy párhuzamos szálon történő végrehajtás idézett elő. A probléma abban merült ki, hogy Sarah válaszát is a FIFO struktúrájú sorba helyezte a rendszer, ezáltal az is értelmezésre és végrehajtásra került és végtelen ciklusokat is képes volt generálni. Megoldásra szolgált ilyen esetben egy szemaforhoz hasonló megoldás, ami blokkol. Ez A „SpeechRecognition” könyvtár fejezet 7. bekezdésében van taglalva. A státusz ablak megjelenítésével és a felhasználó hangbevitelének szabályozása jelentette a megoldást a problémára.

Rögtön a GUI elkészítését követően és Sarah hangjának összeecsatolását követően olyan problémába ütköztem, hogy nem működött a rendszer. Hosszas informálódást követően jöttem rá arra, hogy a Qt esetében a főszálon fut maga a grafikus interfész fut a főszálon, a mellékfeladatok egyéb QThreadekre váltásával megszűnt a probléma.

Hasonló szálak probléma volt A felhasználói felület alcímű alfejezet 6. paragrafiséban taglalt Slit/Signal mechanizmus, pontosabban annak a hiánya. Először a munkásszálak által módosítottam a különféle widgeteket, ez ahhoz vezetett, hogy ezen widgetek kifagytak egy időt követően. Az informálódás után jöttem rá arra, hogy alkalmazni kell ezt a mechanizmust ahhoz, hogy az alkalmazás felhasználói felülete tökéletesen működjön.

A kriptográfia kapcsán is akadt némi probléma, ennek az orvoslásaképpen jött létre a myKey.key file. A Python által támogatott Fernet könyvtárt használva minden futtatásnál új kulcs került generálásra, A Fernet módszernél taglalva volt ennek a kivitelezése. A probléma ott adódott, hogy a szimmetrikus titkosítás miatt képtelen volt visszafejteni az adatokat a rendszer.

6. Következtetések

6.1.Megvalósítások

A dolgozat egy virtuális asszisztenst mutatott be, annak működését írta le. Ennek főbb komponensei a hangfelismerés, hangfeldolgozás, a kriptográfia, a különféle parancs végrehajtó egységek, illetve maga a felhasználói felület, amely egy asztali alkalmazás.

Sikerült több szálak programozás segítségével megalkotni egy egyszerre hallgató és végrehajtó asszisztenst, amelyre később az asztali alkalmazásszerű felhasználói felületet rá lehetett húzni.

A felhasználóbarát stílusok által sikerült egy letisztult, modernnek mondható felhasználói felületet megalkotni, amelyet bármilyen felhasználó könnyedén tud kezelni, és az alkalmazás által nyújtott lehetőségeket egyszerűen megtalálni. Ezen design teljesen átlátható és modern, ezáltal könnyedén eladható.

Az architektúra könnyedén kibővíthető egyéb elemekkel, ha a funkcionalitások kiterjesztése éppen ezt kívánná. A funkcionalitások bővítése rendkívül egyszerűen megoldható, implementálható ugyanúgy, mint a különféle egyéb ablak létrehozása a Qt technológiában.

A kriptált adatokat tartalmazó config fájl teljesen biztonságossá teszi a rendszert adatvédelmi szempontból, ez egy nagyon fontos megkötés volt a rendszer tervezésekor, ugyanis manapság ez a téma rendkívül érzékeny.

Összességében elmondható, hogy Sarah részekre lebontva megfelel egy virtuális asszisztensnek, amely a mindennapokban megkönnyítheti a felhasználója dolgát könnyen elvégezhető, ám gyakori feladatok elvégzésével, ezzel időt spórolva, hatékonyabbá téve a munkát.

6.2. További fejlesztési irányok

Sarah, illetve egy virtuális asszisztens esetében a további funkciók bevezetése folyamatos fejlesztési lehetőségeket nyújt a fejlesztője számára. Esetemben is az egyik legfőbb terjeszkedési irány is ez projekten belül, minél több funkcionalitás bevitele, ugyanis ettől lesz okosabb az asszisztens és ezáltal hasznosabb is komplettebb is.

A platformfüggetlenség is egy hasonló továbbfejlesztési lehetőség, ezzel széleskörűbbé lehetne tenni Sarah elérhetőségét, akár mobilos applikációk, vagy egy kicsit megreformáltabb weboldal segítségével. A platformfüggetlenség kvalitásbéli változtatásokat követelne és vonna maga után, több jelenlegi opció kihalna, azonban újak léphetnének a helyükbe.

A funkcionalitásokon túl konkretizálni lehetne a rendszert, tehát nem szöveg alapú keresés és végrehajtás lehetne az alap, hanem gyakori konkrétabb kérdésekre keresne választ, ilyen például az, hogy mikor zár egy konkrét nagy üzletlánc a közelben, illetve, hogy mikor jár le az autónak a kötelező biztosítása. Ezen szemantika alapú keresés hatalmas fejlesztési lehetőség és egy nagyon jó irányt mutathat.

Végül de nem utolsó sorban a kérdőívet kitöltőkre hivatkozva szeretnék lehetőséget adni a felhasználók számára, hogy tesztre tudják szabni az alkalmazás design-ját, ugyanis rendkívül megosztó volt a sötét téma. Ennek kiküszöbölése megoldható lenne néhány különböző megjelenítési beállítás üzembe helyezésével.

7. Irodalomjegyzék

- [1] A. S. Tulshan, „Survey on Virtual Assistant: Google Assistant, Siri, Cortana, Alexa,” Bangalore, 2018.
- [2] A. Statement, „Improving Siri’s privacy protections,” 28 8 2019. [Online]. Available: <https://www.apple.com/in/newsroom/2019/08/improving-siris-privacy-protections/>. [Hozzáférés dátuma: 20 10 2020].
- [3] S. Tsoukalas, „How does mycroft work?,” 14 11 2018. [Online]. Available: <https://mycroft.ai/blog/guest-blog-hey-mycroft-how-do-you-work/>. [Hozzáférés dátuma: 21 10 2020].
- [4] M. Kapko, „Cortana explained: How to use Microsoft's virtual assistant for business,” Microsoft, 7 2 2018. [Online]. Available: <https://www.computerworld.com/article/3252218/cortana-explained-why-microsofts-virtual-assistant-is-wired-for-business.html>. [Hozzáférés dátuma: 22 10 2020].
- [5] L. Ficsor, Z. Krizsán és P. Mileff, Szoftverfejlesztés, Budapest: NIIF, 2004.
- [6] M. Summerfield, „Rich Text and Printing,” in *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*, Hoboken, Prentice Hall, 2007, pp. 381-412.
- [7] M. Summerfield, „Multithreading,” in *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*, Hoboken, Prentice Hall, 2007, pp. 357-559.
- [8] M. Summerfield, „Signals and Slots,” in *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*, Hoboken, Prentice Hall, 2007, pp. 127-137.
- [9] M. Nilsson és M. Einarsson, Speech Recognition using Hidden Markov Model, Karlshamn: Kaserntryckeriet AB, 2002.
- [10] D. Amos, „The Ultimate Guide To Speech Recognition With Python,” Real Python, 12 5 2018. [Online]. Available: <https://realpython.com/python-speech-recognition/>. [Hozzáférés dátuma: 10 12 2020].
- [11] I. El Gaabouri, C. Asaad és R. Naoufal, *Fernet symmetric encryption method to gather mqtt e2e secure communications for IoT devices*, 2020.
- [12] L. Buttyán és I. Vajda, „Szimmetrikus kulcsú blokkrejtjelezők,” in *Kriptográfia és alkalmazásai*, Budapest, Typotex, 1012, pp. 25-75.

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE, TÎRGU-MUREȘ
SPECIALIZAREA CALCULATOARE

Vizat decan

Conf. dr. ing. Domokos József

Vizat director departament

Ș.l. dr. ing Szabó László Zsolt