ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

[redacted]

OE-NIK	Hallgató neve:	**Kuklis Benjamin**
2022	Hallgató törzskönyvi száma:	**T/006013/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Kuklis Benjamin**
Törzskönyvi száma: T/006013/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Digitális személyi asszisztens megvalósítása
Building a digital personal assistant

Intézményi konzulens: Balázs Elemér
Külső konzulens:

Beadási határidő: 2021. december 15.

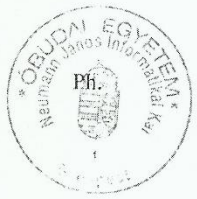
A záróvizsga tárgyai: Számítógép architektúrák
Big Data és üzleti intelligencia specializáció

A feladat

A hallgató feladata egy olyan digitális személyi asszisztens tervezése és megvalósítása, mely architektúráját tekintve moduláris felépítést követ, ezáltal könnyen bővíthető új funkcionalitásokkal. A személyi asszisztens legyen képes alapvető interakciókra, és az idő előrehaladtával ismerje ki a felhasználó szokásait. A felhasználó és a program közötti kommunikáció egy chatbot segítségével valósuljon meg. Az alkalmazás kezdetben két modullal rendelkezzen: hír elemző, valamint napi rutin támogató egységgel.
A hír elemző modul középpontjában a felhasználó tájékozódásának segítése, a napi történések, hírek megismerése álljon. A program internetes forrás (RSS) felhasználásával legyen képes az adatokat kategóriákba (politika, bulvár, üzlet, sport, időjárás, tőzsde stb.) sorolni és annak tartalmi lényegét összefoglalni majd a felhasználó felé kommunikálni.
A napi rutin támogatása során a személyi asszisztensnek tudnia kell előre definiált rutinokat megvalósítania a felhasználó kérésére, mint például a hálózat tesztelése, fejlesztői munkakörnyezet beállítása, weboldal megnyitása, hátralévő feladatok lekérdezése, határidők figyelmeztetése stb.

A dolgozatnak tartalmaznia kell:

- a feladat leírását,
- a kapcsolódó irodalom részletes feldolgozását,
- hasonló rendszerek bemutatását és jellemzését,
- az alkalmazás teljeskörű tervezését és a kiválasztott technológiák indoklását,
- az aktuális hírek letöltéséhez, feldolgozásához és csoportosításához készített saját komponenst, amely a híreket feldolgozza és egy adatbázisban eltárolja,
- a napi rutint támogató modult,
- az alkalmazás tesztelési jegyzőkönyvét és eredmények bemutatását,
- továbbfejlesztési lehetőségeket,
- a dokumentációt, a programot, a szükséges inputadatokat, valamint a rendszert bemutató honlapot és prezentációt CD mellékleten.



.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2023. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
7. sz. melléklet

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022.05.10.

.....
hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: KUKLIS BENJAMIN

Neptun kód: [redacted]

Tagozat: ESTI

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

DIGITÁLIS SZEMÉLYI ASSZISZTENS MEGVALÓSÍTÁSA

Szakdolgozat / Diplomamunka[2] címe angolul:

BUILDING A DIGITAL PERSONAL ASSISTANT

Intézményi konzulens: BALÁZS ELEMÉR

Külső konzulens: —

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.02.11.	PROJEKT TÉMÁJÁNAK KIDOLGOZÁSA	
2.	2021.03.18.	IRODALOMKUTATÁS ÁTTEKINTÉSE	
3.	2021.04.13.	HASONLÓ RENDSZEREK MEGBESZÉLÉSE, ÉRTÉKELÉSE	
4.	2021.05.04.	RENDSZERTERV BEMUTATÁSA, VÉLEMÉNYEZÉSE	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2021.05.04.

intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: KUKLIS BENJAMIN

Neptun kód: [redacted]

Tagozat: ESTI

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

DIGITÁLIS SZEMÉLYI ASSZISZTENS MEGVALÓSÍTÁSA

Szakdolgozat / Diplomamunka[2] címe angolul:

BUILDING A DIGITAL PERSONAL ASSISTANT

Intézményi konzulens: BALÁZS ELEMÉR

Külső konzulens: ———

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.27.	EGYEZTETÉS A CHATBOT BEÜZEMELÉSÉRŐL	
2.	2022.03.15.	RSS HÍRFELDOLGOZÓ KIALAKÍTÁSÁNAK MEGBESZÉLÉSE	
3.	2022.05.01.	AZ ELKÉSZÜLT PROGRAM BEMUTATÁSA	
4.	2022.05.10.	TESZTELÉS, LEADANDÓ DOKUMENTUMOK ÁTTEKINTÉSE	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022.05.10.

intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

TARTALOMJEGYZÉK

1. BEVEZETÉS

Az ember és gép közötti kommunikáció az informatika egy folyamatosan fejlődő területe, az elmúlt években a digitális személyi asszisztensek térhódítása egy merőben új lehetőséget biztosít az eddigi megszokott számítógépes interakciók területén. Ma már a technológiai innovációknak köszönhetően nem elképzelhetetlen, hogy a számítógép úgy segítse az emberek életét, hogy a felhasználóval interakcióba lépve, egy ember és ember közötti beszélgetést leutánozva tegye azt meg. Ez forradalmasítja a felhasználói élményt és új kapukat nyit meg azáltal, hogy nem a megszokott grafikus felületet böngészve jutunk információhoz vagy végezzük el a mindennapi teendőinket, hanem mintha egy baráttal chatelve vagy akár a hangvezérlés segítségével beszélgetve kapnánk instrukciókat.

A mai digitális személyi asszisztensek már elég kiforrott funkcionalitásokkal rendelkeznek ahhoz, hogy ezt gördülékenyen le lehessen bonyolítani. Persze a sci-fikben bemutatott utópisztikus álomvilágtól még elmaradottnak tűnhetnek a mai intelligens asszisztensek képességei, azért a többségük már emberi nyelven képes olyan parancsokat értelmezni és azokat végrehajtani, mint például: *szükségem van-e esernyőre a mai nap?, mikor játszák a várva-várt filmet a moziban?, milyen eredménnyel ért véget a tegnap esti meccs?, mikor indul a következő vonat?* Ezekre a kérdésekre ma már nyugodtan várhat választ a felhasználó, de olyan parancsok is kiadhatóak, amelyekkel a használatban lévő eszköz funkcionalitásaihoz férünk hozzá hangvezérlés által, mint például üzenetküldés az általunk diktált szöveggel, zene lejátszása a megadott előadótól, bevásárló lista bővítése a kért hozzávalókról a vacsorához, útvonal kérése a legközelebbi benzinkúthoz vagy éttermi asztal foglalása.

Természetesen nem csak a mindennapokban vannak előnyei a digitális személyi asszisztensek használatának. Még ha nem is váltja ki az emberi munkaerőt, az üzleti életben tehermentesítheti például a monoton, folyton ismétlődő kérdések súlya alól azokat az ügyfélszolgálati munkavállalókat, akik csúcsidőben képtelenek a hirtelen megnövekedett keresletet kielégíteni. Ebből kifolyólag a chatbotok térhódítása az üzleti szférában az elmúlt években jelentős, előszeretettel alkalmazzák számos különböző területen, mint például: marketing, ügyfélszolgálati tevékenység szinte bármilyen területen (pl.: szórakoztatóiparban, mozi vagy rendezvények területén segít tájékozódni). A felhasználó által küldött üzenetet értelmezve képesek az adott problémára választ adni, amely óriási előnyt jelent például egy összetett felület megismerésekor vagy értékesítési területeken új termék bevezetésekor sikeresen összeköti a potenciális vásárlót a céggel. Az egészségügyi használata során élő példák között tekinthetjük például azt az esetet, amikor a betegek az adott szolgáltató chatbotján keresztül kapnak információt arról, hogy tüneteik súlyossága alapján szükség van-e professzionális segítségre vagy akár saját maguk is meg tudják oldani a problémát. A betegek emlékeztetése a gyógyszereik bevételére, az állapotuk monitorozása [33]. Mindemellett még a játékpiacon is előfordulnak chatbot megvalósítások [34].

2. SZAKIRODALMI ÁTTEKINTÉS

Az alábbi fejezetben a digitális személyi asszisztens fogalmának bemutatása, a párbeszéd modellezési lehetőségek felkutatása és részletezése, a beszélgető robotok korai megvalósításaiból néhány kiemelkedő példa áttekintése, a modern szemlélet vizsgálata a chatbotok területén a beszélgetés modellezése és technológiai megvalósítás szempontjából, valamint a hasonló rendszerek áttekintése és bemutatása valósult meg a releváns szakirodalmak felkutatásával.

2.1. Digitális személyi asszisztens fogalom bemutatása

A szakirodalom számos eltérő elnevezést használ a digitális személyi asszisztensre, ezek többek között virtuális asszisztens (angolul *Virtual Assistant*, röviden *VA*), intelligens (személyi) asszisztens (angolul *Intelligent Personal Assistant*, röviden *IPA*), Csevegő robot (angolul *Conversational Agents*), hangvezérléses robot (angolul *Voice Controlled Agents*) és még megannyi elnevezés. A terminológia sokszínűségének ellenére azonban a lefedett funkcionalitások köre megegyezik. A digitális személyi asszisztens egy olyan szoftver, amely akár saját, dedikált eszközön (például *az Amazon Echo, Google Dot*), vagy okoseszközökbe integráltan, vagy akár számítógépen fut (például *Microsoft Cortana*). Tervezésük során kihangsúlyozott szerepet kap, hogy beszéd és írás felismerésére alkalmas legyen és a felhasználó kéréseire természetes nyelvfeldolgozás segítségével emberi kommunikációt szimulálva valós időben válaszoljon. Valamint képes *IoT* eszközök vezérlésére és egyéb feladatok elvégzésére is alkalmas. Működésük elengedhetetlen feltétele az internet kapcsolat, amelynek segítségével a szükséges adatok letöltésre kerülnek a szerverekről és a felhasználói kérések feldolgozásra kerülnek [1].

Az intelligens személyi asszisztensek fejlődésének alapfeltétele az elmúlt évtizedben több területen tapasztalt technológiai fejlődés. Ahhoz, hogy ma a piacon található legyen a való életben is működő megvalósítás a mesterséges intelligencia (angolul *Artifical Intelligence,* röviden *AI*), a gépi tanulás (angolul *Machine Learning*, röviden *ML*) és természetes nyelv felismerés (angolul *Natural Language Processing*, röviden *NLP*), az *IoT* és az adattudományok témakörökben volt szükség jelentős előrelépésekre. Ezekben a kérdéskörökben tapasztalt fellendülés segítette elő a virtuális asszisztensek mindennapi alkalmazhatóságát és emelték a szegmens jelentőségét olyan szintre, hogy az a felhasználói réteg számára hozzáadott értékkel szolgálhasson. Ahogy az már említésre került, jelenleg is számos piacon elérhető megvalósítás létezik általános célokra, ezek lehetnek platformspecifikus vagy olyan megvalósítások, amelyek csak egy adott operációs rendszeren képesek működni, mint például a nagy tech óriásvállalatok saját alkotásai az *Apple Siri, Microsoft Cortana, Google Now, vagy az Amazon Alexa* [2].

Azonban léteznek olyan virtuális asszisztensek is, amelyek egy adott feladat elvégzésre specializálódtak. Ilyen például az *Eco-Driving Assistant* témakörben fejlesztett *Artemisa,* ami egy olyan vezetői asszisztens, amely a felhasználó szokásait figyelembe véve ad

tanácsokat a vezetés hatékonyságának növelésére [3]. Vagy egy másik érdekes példa lehet még *Lucy* a táplálkozási szakértő asszisztens, ami az étkezési szokások ellenőrzésével tud a bevitt kalóriamennyiség optimalizálásával a túlsúlyos felhasználók fogyásában segíteni [4]. Továbbá tekinthetjük példaként akár az *Effective Classroom Assistant*-et is, ami tantermi asszisztensként fordítható és a célja, hogy a tanulók új képességeket sajátíthassanak el úgy, hogy a rendszer képes a felhasználónak személyre szabott tanulástechnikai módszert biztosítani azáltal, hogy a megfelelő komplexitású feladatokat adja és visszajelez, kijavítja a hibáit így biztosítva, hogy a felhasználó a saját tempójában haladva dolgozza fel az adott anyagrészt [5].

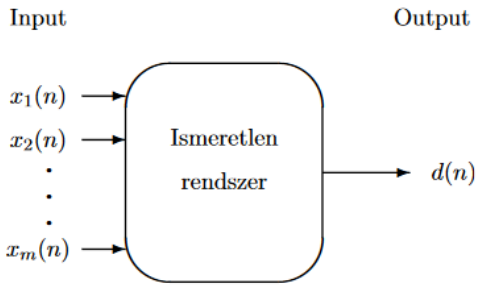
2.2. Párbeszéd modellezési szemléletek

A felhasználó számítógép közötti kommunikáció modellezésére két főbb megközelítés létezik. Az egyik az, amikor a felhasználótól érkező üzenetben mintákat keresve, előre definiált szabályok alapján, előre rögzített válaszokat keres. Ez a chatbotok születésekor megalkotott kezdetleges megvalósítás. Manapság *deep-learning* alapú megoldásokat használnak a hasonló rendszerek. A *deep-learning* egy neurális hálón alapuló technológia, amelyben nagy mennyiségű adat segítségével tanítható meg a rendszer arra, hogy a bemeneti kifejezésekre értelmes és nyelvtanilag is helyes válaszokat adjon [15].

2.2.1. Neurális hálózat definíciója, működése

A neurális hálózat a gyakorlatban széleskörűen alkalmazott leginkább tudományos és műszaki feladatok megoldására. Többek között karakterfelismerésre, képfeldolgozásra, jelfeldolgozásra, adatbányászatra, méréstechnikai és szabályozástechnikai feladatokra használnak különböző neurális hálózatokat. Az ilyen jellegű problémák megoldása általában törvényekre, modellekre támaszkodik, mint például a tömegvonzás törvénye vagy az atomok felépítésének modellje. Viszont vannak olyan esetek, amikor nincs korábban felvázolt modell, nincs rendelkezésre álló törvényszerűség, de vannak rendelkezésre álló adatok, megfigyelések, amelyekből következtethetünk valamilyen összefüggésre [16].

A neurális hálózatnak nem célja a jelenség modellezése vagy törvényszerűségek megállapítása, a jelenséget fekete dobozként kezeli, csak a bemenő (*input*) és a kimenő (*output*) adatokat tekinti (*2.2.1. ábra*) [16].



2.2.1. ábra A modellezendő rendszer [16]

A neurális hálózat neuronok összekapcsolt csoportja, amely a biológiai neurális hálózat néhány tulajdonságát modellezi. „*A mesterséges neurális hálózat az idegrendszer felépítése és működése analógiájára kialakított számítási mechanizmus*” [16]. A természetes és mesterséges neurális hálók működése többnyire megegyezik, ezen hálózatok alapelve ugyanis az egymással összekapcsolt feldolgozóegységekben - neuron - rejlik, amelyek a műveleteket végzik. A neurális háló a neuronok rendezett topológiájú nagymértékben összekapcsolt rendszeréből áll. A hálóban lévő neuronok információt tárolnak, amely információfeldolgozás módját hivatott az úgynevezett tanulási algoritmus (*learning algorithm*) definiálni. Illetve rendelkezik még a megtanult információ előhívását elősegítő algoritmussal, amit előhívási algoritmusnak (*recall algorithm*) neveznek [16].

A neurális hálózatok működési fázisa szerint két szakaszt különböztetünk meg. Az első fázis, melyet tanulási fázisnak nevezünk, a hálózat kialakítására szolgál, melynek során a hálózatba valamilyen módon beépítjük, eltároljuk a rendelkezésre álló mintákban rejtve meglévő információt. Eredményként egy információ-feldolgozó rendszert kapunk, melynek használatára általában a második fázisban, az előhívási fázisban kerül sor. A két fázis a legtöbb esetben időben szétválik. A tanulási fázis rendszerint lassú, hosszú iterációkat, esetleg sikertelen tanulási szakaszokat is hordoz. Ezzel szemben az előhívási fázis tipikusan gyors feldolgozást jelent [16].

2.2.2. A neurális hálózat elemei, topológiája

Többféle modell létezik, amely manapság számítógéppel modellezhető, mint például a többrétegű perceptron (*MLP*), a radiális alapfüggvényekből álló hálózat (*RVS*) vagy a tartó vektor gép (*SVM*). A neurális hálózatok ma használt talán legismertebb modellje az úgynevezett *MLP* (*Multi Layer Perceptron*) többrétegű perceptron modell. Ebben a modellben három réteget különböztetünk meg (*2.2.2. ábra*):

1. Bemeneti réteg (*input layer*)

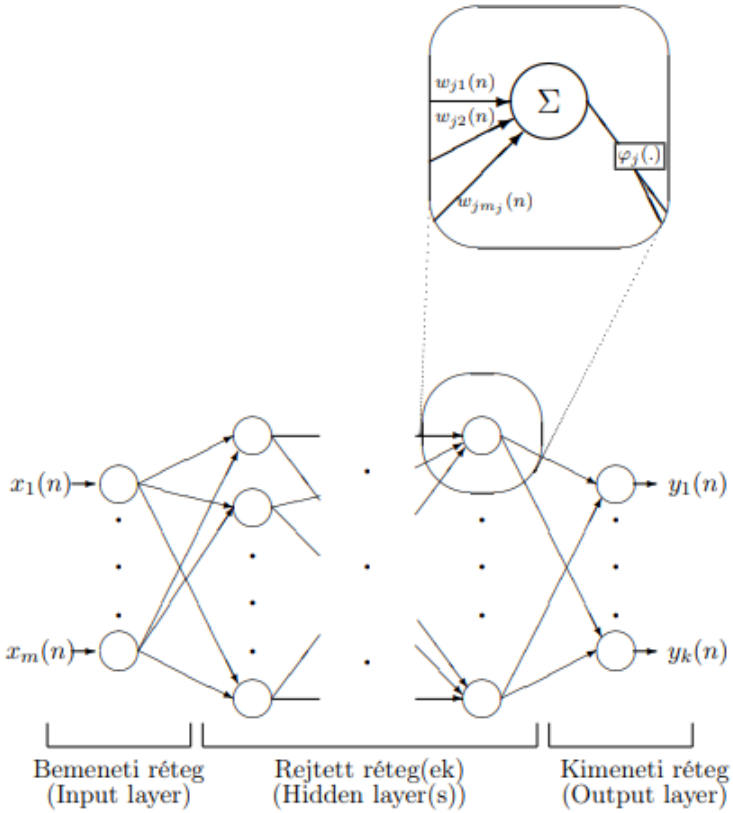
A bemeneti réteg, ahogy elnevezése is sugallja a bemeneti adatréteg, amely numerikus adat bevitelét engedélyezi a rendszerbe, a hálózatok implementálása során talán a legnagyobb időráfordítást igénylő feladat az úgynevezett *data-processing* vagyis az adat oly módon történő manipulálása, hogy az numerikus formában kerülhessen a rendszerbe.

2. Rejtett réteg (*hidden layer*)

A rejtett rétegben - vagy rétegekben - található a legtöbb neuron, ez a réteg felel az adat átalakításáért, hogy a várt eredmény jelenjen meg a kimeneten. Ebben a rétegben jelennek meg a súlyok és torzítások, amiről a későbbiekben lesz még szó a neuron felépítésénél. Ahogy korábban is említésre került, a neurális háló a jelenséget „fekete dobozként" tekinti, a rejtett réteg tulajdonképpen a logika, ami a bemenettől a rejtett rétegen át a kimenet teszi, a fejlesztők a bemeneti és kimeneti réteget használják munkájuk során közvetlenül.

3. Kimeneti réteg (*output layer*)

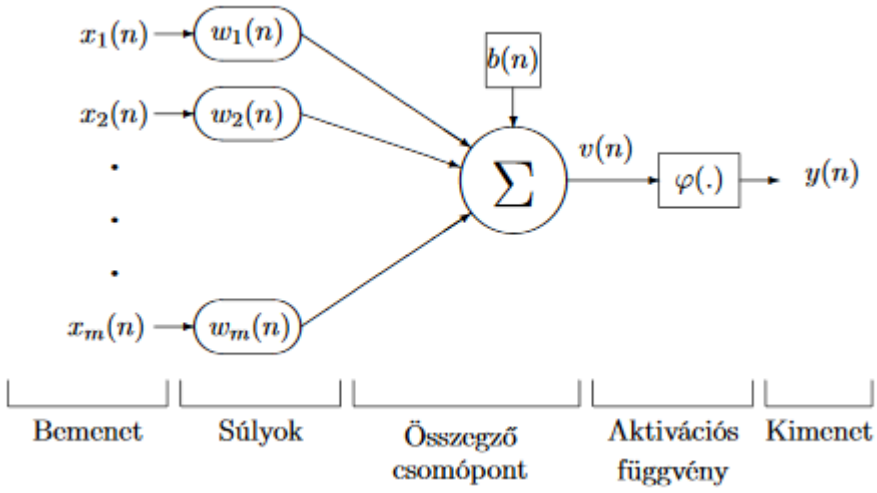
A kimeneti rétegen jelenik meg a végeredmény az adat manipulációját követően, itt megjelenhet egy vagy több neuron is. A környezet felé továbbítja a kívánt információt, típusukra nézve nem feltétlenül különböznek a többi neurontól.



2.2.2. ábra MLP modell topológiája [16]

Az *MLP* modell úgynevezett előrecsatolt hálózat (*feedforward network*), amely annyit jelent, hogy a hálózatba betöltött adatokon műveleteket végezve tovább halad a hálózaton belül míg a kimeneti rétegbe ér. Ahhoz, hogy bármilyen a bemeneten megjelenő adat a kimeneten az elvárt eredményt produkálja a hálózatot tanítani szükséges. A neuronok összeköttetési rendszere alapján megkülönböztetünk visszacsatolt hálózatokat is (*recurrent network*) [16].

2.2.3. A neuron felépítése



2.2.3. ábra A neuron felépítése [16]

A *2.2.3.ábra* egy neuron felépítését mutatja be, ez alapján a következő részeket érdemes kiemelni:

1. Bemenet (*input*)

A bemenet m-dimenziós vektorok $(x_1(n), x_2(n), \dots, x_m(n))$, ahol n=1,2,.. az időpillanatokat jelenti, ezek ismertek számunkra.

2. Súlyok (*weight*)

Az igazi w_1, w_2, …, w_m súlyok nem ismertek, ezek meghatározása a cél. A $w_1(n), \dots, w_2(n), \dots, w_m(n)$ súlyok, amelyeket az n-edik időpillanatban használtunk, ezeket lépésenként finomítjuk.

3. Torzítás (*bias*)

A torzítás sem ismert, ennek meghatározása is cél, ez egy skalár mennyiség $(b(n))$.

4. Összegző csomópont

Ez a bemenet adatainak a súlyozott összege az *(1) képleten* látható.

$$v(n) = b(n) + \sum_{i=1}^{m} w_i(n)x_i(n) \quad (1)$$

5. Aktivációs vagy transzfer függvény

A transzfer függvény az előző pontban meghatározott *v(n)* összegzett értékét alakítja át, értéke pedig a feladat szempontjából alkalmas intervallumba esik. Ezt nekünk kell meghatároznunk.

6. Kimenet (output)

Ez a neuron által a bemeneti $x(n)$ értékhez rendelt érték a *(2) képleten* látható.

$$y(n) = \varphi(v(n)) \quad (2)$$

A neurális hálózatok használatának fő szakaszai:

- A séma megadása: a hálózat sémájának definiálása a perceptron modellezéséből és aktivációs függvény választásából áll,
- A hálózat betanítása: a tanítási pontok meghatározása az input értékek és a hozzá tartozó kimeneti értékek, tulajdonképpen a neuron az inputhoz kiszámolt output értéket hasonlítja a tényleges megadott kimeneti adattal, ez a hálózat hibája. Ezt a súlyok változtatásával lehet minimalizálni,
- A hálózat használata: amennyiben a betanítás sikeres volt a hálózat használható számára ismeretlen értékekkel is [16].

2.3. Korai megvalósítások

Az intelligens asszisztensek történetének áttekintésekor fontos megemlíteni azokat a korai beszélgető robotokat, amelyek először keltették annak a látszatát, hogy a számítógép és ember közötti kommunikáció párbeszédszerűen zajlik. Annak ellenőrzése, hogy számítógép képes-e megtéveszteni válaszaival egy embert, már 1950-ben is foglalkoztatta a számítógéptudományt, amikor is Alan Turing kidolgozta az azóta csak Turing-teszt néven elhíresült módszert. Ennek a tesztnek a felhasználó meggyőzése áll a középpontjában, arról, hogy a beszélgetőpartnere egy ember. Amennyiben a kérdező 5 perc társalgás után sem képes eldönteni, hogy a másik fél gép-e, akkor kimondható, hogy a program sikeresen teljesítette a Turing-tesztet [6]. A Turing-tesztet a Loebner-díj keretében szervezik meg, ahol a legemberszerűbb chatbot kerül díjazásra. A Turing-tesztet mai napig nem abszolválta egyetlen mesterséges intelligencia sem [12].

2.3.1. Eliza

Az első ilyen nyilvánosan elérhető programot 1966-ban alkotta Joseph Weizenbaum és Elizának nevezte el, a chatbot egy terapeuta szerepét szimulálta egy klinikai kezelésben,

távolságtartó és nem rendelkezett egyedi személyiségi jegyekkel [10]. A mesterséges intelligencia látszatát keltette, azonban valójában nem értette meg az üzeneteket csupán azok feldolgozásával volt képes mintákat felismerni a szövegben és előre definiált szabályok alapján választ adni. Akkoriban ez a módszer közel állt a Turing-teszt teljesítéséhez, és sokakat meggyőzött arról, hogy az emberi kognitív folyamatokat leutánozva képes gondolkodni, azonban ennél a megvalósítás jóval egyszerűbb volt [7] [8]. Még az alkotója is meglepődött saját szoftvere sikerén, olyannyira, hogy egyik publikációját arra szentelte, hogy tájékoztassa az embereket, mire képesek és nem képesek a számítógépek és, hogy tulajdonképpen *Eliza* nem jelent akkora áttörést a tudományok területén, mint ahogyan azt a sikere alapján a legtöbb felhasználó gondolta [7] [9].

Eliza megvalósításának kulcsa abban rejlett, hogy a program a kapott üzenetben egyezéseket keresett, amelyekhez előre társított válaszok voltak eltárolva, ezek után annyi dolga volt a programnak, hogy az eltárolt választ a megfelelő szövegkörnyezethez alakítva (helyettesítő karakterek segítségével azonosítva) térjen vissza a válasszal a felhasználóhoz. Az 1. Chatbot társalgás példát tekintve az említett működés bemutatható, azonban csak angol nyelven demonstrálható ez igazán, ezért nem került lefordításra magyar nyelvre. A szövegben olyan karaktert vagy karakterláncot keres, amelyhez van eltárolt válasz, így például a *You are* az *1.Chatbot társalgás példában* egyértelműen behelyettesíthető abba a mintába, ahol tulajdonképpen a személyes névmást kell a környezethez igazítani, illetve az éppen aktuális szövegkörnyezetben lévő mondanivalót hozzáilleszteni az eltárolt mintához kapcsolódó kérdéshez, ezáltal olyan kérdést tett fel a program, ami a beszélgetés kontextusához illeszkedik.

Felhasználó: *You are entitled to your opinion.* *Eliza*: *What makes you think I am entitled to my opinion?*

1. Chatbot társalgás példa - Eliza szöveg minta felismerés

Felhasználó: Jogod van a véleményedhez.
Eliza: Miből gondolod, hogy jogom van a véleményemhez?[10]

Viszont ebből is egyértelműen látható, hogy a megvalósítás során a program nem értelmezi a bemenetet, ennek ellenére *Eliza* mesterséges intelligencia területén az egyik legismertebb chatbottá vált és számos utódjának az alapja lett a későbbiekben [10]. Mauldin chatbotokat összehasonlító írásában fejtette ki melyek azok a vonások, amelyek az *Eliza*val folytatott beszélgetéseket valódivá teszik:

- **kérdések használata** ahhoz, hogy a beszélgetés irányát a felhasználó határozza meg anélkül, hogy a programnak hozzá kellene járulnia a téma felajánlásához. Ez azért is volt fontos, mert a chatbot témájába beleillik az, hogy a beteget végig hallgatja a terapeuta, és nem vág közbe új témákkal,
- **a felhasználó által használt szavak felhasználása a kérdésekben** (1. Chatbot társalgás példa),

- **Rogers személyközpontú terápia,** lényege, hogy a felhasználó van a központban és a program nem szakítja félbe saját kérdéseivel, elkerüli a deklaratív kijelentéseket, annak érdekében, hogy a kommunikáció során ne hozza magát ellentmondásos helyzetbe.

2.3.2. Parry

Az *Eliza* chatbothoz nagyon hasonló felépítéssel rendelkező szoftver a *Parry*. 1971-ben jelent meg, Kenneth Colby által. *Parry* elsősorban abban különbözött *Eliza*-tól, hogy egy olyan paranoiás beteget személyesít meg, aki megosztja a saját félelmeit a beszélgetőpartnerével. *Eliza*val szemben *Parry* azért számít maradandó alkotásnak, mert a paranoiás állapotot szimulálva olyan jellemvonásokat, egyedi beszélgetési szokásokat ad a programnak, amely saját személyiséget tükröznek a beszélgetés során. A két chatbot között azonban a legnagyobb különbséget az jelentette, hogy *Parry* már rendelkezett egy saját „elmeállapottal" és képes volt a beszélgetés előzményeit is megtartani. Ezek alapján egy beérkező üzenet feldolgozása már nem csak abból állt, hogy a bemenetre érkező szöveget feldolgozza, hanem *Parry* saját szándéka, nézete és véleménye is számításba került [10]. Mauldin írásában jól összefoglalta, melyek azok a trükkök, amelyek *Parry* kommunikációját még emberszerűbb beszélgetőtárssá teszik:

- **elismeri, ha egy témához nincs válasza**, egy olyan kérdésre, amelyre nem tud válaszolni, egyszerűen „*nem tudom*" a válasz,
- **meg tudja változtatni a beszélgetés témáját,**
- **rövid történetek közöl a maffiával kapcsolatban,** ezzel megváltoztatja a témát és ragaszkodik ahhoz, hogy befejezze [10] [11].

```
Parry: I don't get you.
...
Parry: Let's talk about something else.
...
Parry: I know the mob controls the big rackets.
```

2. Chatbot társalgás példa - Parry beszélgetés trükkjei
Parry: Nem értelek.
Parry: Beszéljünk valami másról.
Parry: Tudom, hogy a maffia irányítja a szervezett bűnözést. [10]

A legfontosabb kritika, amit a program kapott akkoriban az volt, hogy *Parry* nem több, mint egy illúzió, csak a látszatát kelti annak, hogy valós gondolatokkal rendelkező emberi kommunikációt produkál, valójában nem képes rá. *Parry* alkotója, Kenneth Colby akkoriban így válaszolt a kritikára: „*Egy paranoiás beteg modellezése a következő tényezők figyelembevételével valósítható meg: egyszerre kell paranoiásnak, betegnek és emberinek hatnia, az első kettőben Parry kimondottan jól teljesít, de a harmadikban megbukik, mert nem rendelkezik elég tudással. Parry nem egy igazi ember, csak egy modell, egy szimuláció, egy utánzás, egy észszerű készítmény, egy automatizáció, szintetikus és mesterséges."* [10].

2.3.3. A.L.I.C.E.

Richard Wallace 1995-ben készítette el *A.L.I.C.E.* nevű chatbotját, ami két alkalommal is elnyerte a Loebner-díjat. A megvalósítása nagyon hasonló ez *Eliza* által használt mintakereső megoldásra, a fő cél a megvalósítása során az volt, hogy minél tovább tudjon úgy kommunikálni a felhasználóval, hogy számára felismerhető lenne, hogy egy géppel folytat beszélgetést. *A.L.I.C.E.* nem rendelkezik speciális tematikával, egy általános emberi kommunikáció modellezése volt a célkitűzés. Az egyik legjelentősebb újítás, ami *A.L.I.C.E.* sikerét okozta az *AIML* azaz, *Artifical Intelligence Markup Language,* ami egy olyan jelölőnyelv, ami az XML kiterjesztésének számít. Tulajdonképpen az *Eliza* mintafelismerésének a továbbfejlesztése, hiszen itt is a bemenet alapján keres rá az eltárolt minták között arra, hogy van-e ehhez kapcsolódó eltárolt válasz. Az `<AIML>` jelölő segítségével lehet megadni a fájl típusát, ez a gyökér elem, a `<TOPIC>` egy kiemelt jelentőségű, opcionális elem, amelyhez a témát jelöli és ehhez tartozhat több kategória, amit a `<CATEGORY>` elem jelöl. A kategória a tudás egyik alapegysége, minden kategória tartalmaz egy felhasználói bemenetre illő mintát és egy ahhoz tartozó választ, ezeket a `<PATTERN>`, azaz minta és a `<TEMPLATE>`, azaz a választ tartalmazó elemek jelölik (*2.3.1. ábra*). Az *AIML* mintakeresés legfőbb célja, hogy megtalálja a legjobb és leghosszabb egyezést az eltárolt minták és a felhasználó által beadott inputok között. A technológia használatával korábban még nem látott teljesítményjavulást értek el a természetes nyelvfeldolgozás területén [13] [14].

```
<AIML VERSION=”1.0”>
      <TOPIC NAME=”CASUAL”>
            <CATEGORY>
                  <PATTERN>How are you?</PATTERN>
                  <TEMPLATE>I am fine, and you?</TEMPLATE>
            </CATEGORY>
      </TOPIC>
</AIML>
```

2.3.1. ábra - AIML példa

2.3.4. Jabberwacky és CleverBot

A chatbotok között mindenképp említést érdemel a Rollo Carpenter által megalkotott *Jabberwacky*. Fontos kiemelni, hogy itt a felhasználó szokásainak és szavainak elemzése és megtanulása van a középpontban. Szintén többször elnyerte a Loebner-díjat. A *Jabberwacky* továbbfejlesztése a *CleverBot*, ami szintén Rollo Carpenter nevéhez fűződik. A *CleverBot* talán ma a legismertebb chatbot. Szintén a tanulás alapján ad válaszokat, tehát ha korábban volt már hasonló kérdés feltéve, akkor az ahhoz tartozó választ adja vissza, nem rendelkezik saját, előre megadott válaszokkal [10].

2.4. Modern szemlélet

A neurális hálók használata a természetes nyelv feldolgozása terén a XXI. század nagy áttörése, a neurális hálók hatékony tanítását az 1990-es években nem lehetett megvalósítani a több rejtett réteggel rendelkező hálónál, mert a tudományterületnek további fejlődésre volt szüksége. Ahhoz, hogy új lendületet vehessen a mesterséges intelligencia a nyelvfeldolgozás terén a következő három feltételnek kellett teljesülnie: új tudományos felfedezések, nagyobb számítási teljesítmény, valamint több adat. A tudományos felfedezések, amelyek a legfontosabb előrelépéseket jelentették többek között Yann LeCun a New York-i egyetem professzora, aki a kép- és beszédfelismerés területén ért el áttörést a LeNet elnevezésű konvolúciós neurális hálóval. A ma használt mély tanulás alapú természetes nyelvfeldolgozás alapja az úgynevezett Long Short-Term Memory (LSTM) hálózat elméletének kidolgozása is erre az időszakra tehető. Geoffry E. Hinton nevét pedig a neurális hálózatok tanítása miatt kell megemlíteni, hiszen ő terjesztette ki egy korábban létező, úgynevezett *backpropagation* eljárást, több hálózatra is. Amellyel az alakzatok felismerése és a szövegben következő szó jósolható meg nagyobb pontossággal. A nagyobb számítási teljesítményt a grafikus kártyák fejlődése jelentette, a neurális hálózatok tanítása során ugyanis nagyszámú mátrixművelet végrehajtására van szükség. A big data fejlődése pedig lehetővé tette, hogy nagymennyiségű adattal legyenek a neurális hálók taníthatóak [17][18].

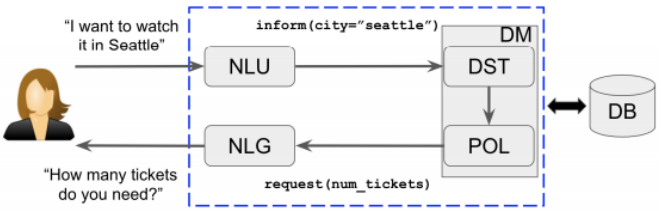
2.4.1. A beszélgetés modellezése

Attól függően, hogy az asszisztens milyen problémát old meg, mi a fő funkciója két típust különböztethetünk meg:

- szociális chatbot, ahol a felhasználó társként tekint a chatbotra és fő hangsúly az érzelmi kapcsolaton van (például: *Microsoft XiaoIce*),
- feladatközpontú chatbot, ennél a típusnál a felhasználó által megadott feladatok elvégzése áll a központban (például*: Amazon Alexa, Apple Siri, Google Home, vagy Microsoft Cortana*) [18].

2.4.1.1. Feladatközpontú chatbot párbeszéd modell

A feladatközpontú chatbotok általában egy külső adatbázisból szereznek a feltett kérdésre választ, a párbeszéd modellezésekor négy fő modult különböztetünk meg (*2.4.1. ábra*) [19][22].



2.4.1. ábra A feladatközpontú chatbot beszélgetés modellezése [19]

Természetes nyelv megértéséért felelő modul (Natural Language Understanding - NLU): a felhasználó által küldött üzenetből ki tudja következtetni, hogy mi a szándéka, mit szeretne elérni, mi a feladat lényege, alapvető információk kivonása.

Párbeszéd menedzser (Dialogue Manager - DM):

A *DM* két további modulra bontható:

- *Párbeszéd állapotának követéséért felelős modul (Dialogue State Tracking - DST)*: a felhasználóval való beszélgetés során a chatbotnak szüksége van egy állapot fenntartására annak érdekében, hogy megőrizhesse a már korábban a párbeszéd során felmerült megfigyeléseit,
- *Párbeszéddel kapcsolatos irányelvekért felelős modul (Dialogue Policy - POL):* a modul feladata, hogy a *DST* által megadott állapot szerint visszatérjen válasszal a rendszer vagy esetleg egyéb műveletet hajtson végre.

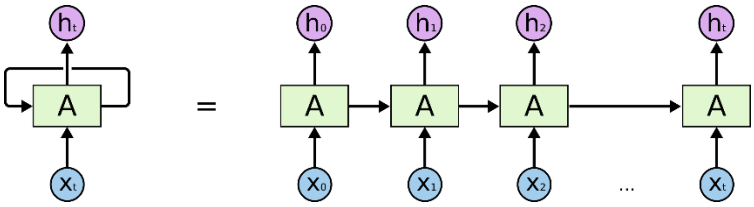
Természetes nyelv generálásáért felelős modul (Natural Language Generation - NLG): ha a *POL* úgy döntött, hogy a felhasználónak üzenetet kell küldeni, akkor ez a modul felel az információ szavakba öntéséért [18][19].

2.4.2. Természetes nyelv feldolgozása neurális hálózattal

A neurális hálózatok használata a természetes nyelvfeldolgozás területén lehetővé teszi, hogy a felhasználó által küldött üzenetből meghatározható legyen a kérésének szándéka. Ennek feldolgozása a mondat szavakra bontásával kezdődik, majd a szavakat a neurális háló bemenetére kell kötni. Itt a szavakat egyesével egy alacsonyabb reprezentációra kell transzformálni ahhoz, hogy számításokat lehessen velük végezni. Az alacsonyabb reprezentáció itt olyan vektorokat jelent, amelyek a szavak jelentését képesek ábrázolni. A szövegfeldolgozás visszacsatolt neurális hálózatok segítségével történik [15].

2.4.3. Visszacsatolt neurális hálózatok

A visszacsatolt neurális hálózat (*Recurrent Neural Network – RNN*) egy olyan hálózat, amely bemenetként képes egy sorozatnyi elemet venni. Erre azért van szükség, mert a felhasználó által megadott szöveg előre nem meghatározott hosszúsággal rendelkezik. Az előrecsatolt hálózatokhoz képest abban különbözik, hogy az előző állapotokat is képes számításba venni a súlyok meghatározásakor az úgynevezett hurkok segítségével (*2.4.2. ábra*). Az előző állapotokra pedig azért van szükség, hogy a szövegből úgy lehessen kivenni a felhasználói szándékot, hogy az összes szó figyelembe legyen véve [20] [21].

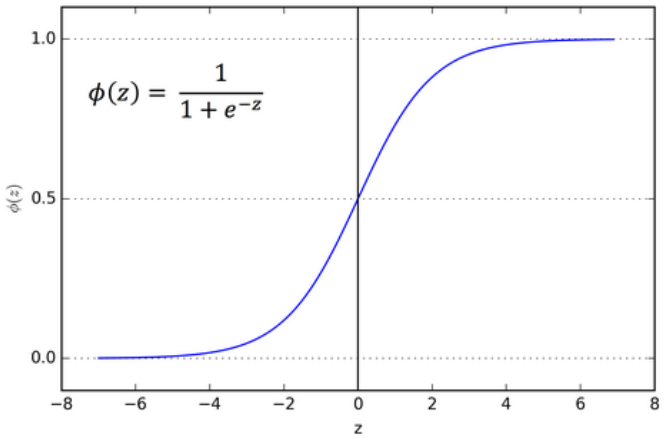


2.4.2. ábra Visszacsatolt neurális hálózat [20]

Azonban a visszacsatolt neurális hálózatoknak a legegyszerűbb megvalósítása ritkán használt megoldás, mert egy esetlegesen hosszabb mondatban a függőségek nem felismerhetőek a modell által. Ez az úgynevezett eltűnő gradiens problémából fakad, ugyanis az alsóbb rétegek súlyai egyszerűen nem tudnak érvényesülni, és ezáltal nem lesz hatásuk a kimentre. A problémára megoldást az *LSTM modell* jelent [20].

2.4.3.1. Az LSTM modell

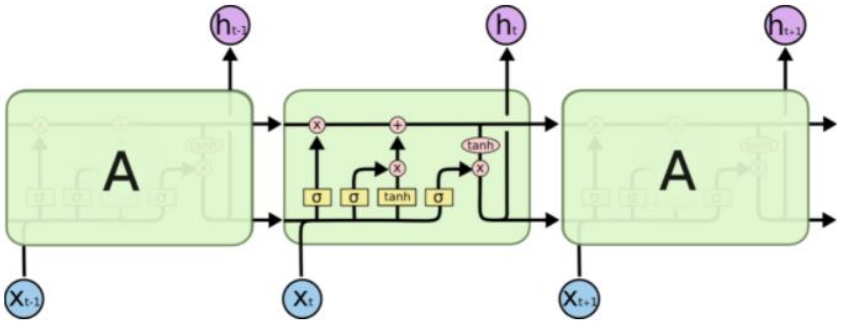
Az *LSTM modell (2.4.4. ábra)* a visszacsatolt neurális hálózatok azon fajtája, amely képes a függőségek felismerésére akár egy hosszabb szövegben is. Felépítésében kulcsfontosságú szerepet kap négy egymással együttműködő komponens, amely a hálózat különlegességét adja. Az egyik legfontosabb úgynevezett cella állapot részegység, amely futószalag-szerűen halad végig az *LSTM* láncon. A cella állapotát törölhetjük vagy hozzáadhatunk információt a feldolgozás során, de az információ akár változatlanul is áthaladhat a következő állapotba. Amennyiben az áthaladó információ változtatására van szükség a hálózatban műveletet kell végezni az adattal, amely a „kapuk" segítségével valósul meg. A kapuk aktivációs függvényeket hajtanak végre, ami lehet szigmoid függvény (*2.4.3. ábra*), hiperbolikus tangens függvény vagy pontonkénti szorzási művelet. A szigmoid függvény értékkészlete 0 és 1 között van, így a szigmoid kapu legfőképpen valószínűség jellemzésére használatos [20][32].



2.4.3. ábra Szigmoid függvény [32]

A hiperbolikus tangens függvény értékkészlete -1 és 1 között van, így ez a függvény alkalmas a negatív értékek feltüntetésére is [20] [32].

Az egyik kulcsfontosságú réteg az úgynevezett *forget gate layer*, vagy magyarul a *felejtésért felelős réteg,* amely szintén egy szigmoid kapuból áll, ez az egység határozza meg, hogy az állapotot elvetjük vagy megtartjuk. A következő réteg az *input gate layer* azaz a bemeneti kapu réteg határozza meg, hogy milyen új információt adunk hozzá a cella állapotához. A bemeneti kapu rétegben egy szigmoid függvény segítségével meghatározásra kerül, hogy milyen új információ lesz hozzáadva, majd egy hiperbolikus tangens függvény a hozzáadandó új információból vektort képez, végül a kettő kombinációja adja azt a frissítést, amelyet az állapothoz adunk. Fontos megjegyezni, hogy a cella állapotának frissítése időben egyszerre jelenti a *forget gate layer* és az *input gate layer* feladatait. Utolsó lépésként a kimenet meghatározása következik, itt szintén egy szigmoid függvény dönti el, hogy mi lesz a kimeneten majd a cella állapot hiperbolikus tangens függvénye és a szigmoid függvény szorzata adja a kimenetet [20].



2.4.4. ábra LSTM modell [20]

2.5. Hasonló rendszerek

Manapság a piacon különbséget tehetünk a digitális személyi asszisztensek között, akik hangalapú vezérléssel valósítják meg a kommunikációt és a chatbot megoldások között, amelyek írott szöveget értelmezve lépnek interakcióba az emberekkel. Amennyiben a szakdolgozat célját tekintjük, ami a hírek feldolgozása, a piac megosztottá válik, ugyanis ezek főként a chatbotok körében népszerű funkcionalitás. Mivel az általam vizionált cél egyedinek számít, a hasonló rendszerek vizsgálatakor kitérek ugyan a két területre, azonban ugyanolyan megvalósításról beszámolni nem tudok.

2.5.1. Hírfeldolgozó chatbotok

Az utóbbi években számos nagy médiacég kezdett saját chatbot fejlesztésbe, amelynek központjában a hírek személyre szabott, érdeklődési körnek megfelelő küldése áll. Azonban a nagyobb közönség elérése érdekében a legnagyobb hírportálok ezeket jellemzően a Facebook Messengerbe integrálva tárják felhasználóik elé, saját asztali alkalmazást nem használnak. A hasonló rendszerek elemzésekor 3 céget választottam, akiknek a funkcionalitásait kielemeztem az interneten megtalálható információk segítségével, ezek a *CNN* az *AlJazeera* és a *FoxNews*. Mivel Európában a szolgáltatásaik nem elérhetőek, ezért csak a terméket bemutató cikkek alapján határoztam meg a főbb funkcionalitásokat (*1.táblázat*).

	CNN	**AlJazeera**	**FoxNews**
Önálló, asztali alkalmazás	nem	nem	nem
Elérhető beépített chatszolgáltatásként (pl.: Facebook Messenger)	igen	igen	igen
Időzített napi értesítések	igen	igen	igen
Személyre szabott hírek	igen	igen	nem
Természetes nyelv feldolgozása (pl.: hír kategória alapján)	igen	igen	nem
Beszédfelismerés	nem	igen	nem
Szöveg összefoglalása	nem	nem	nem

1. táblázat Hírfeldolgozó chatbotok összehasonlítása [22][23][24][25][26]

2.5.2. Digitális személyi asszisztensek

A digitális személyi asszisztensek témakörében már létezik hangalapú megvalósítás és komolyabb támogatás a mindennapi életben felmerülő problémák megoldására is. A következő három jelölt képességeit elemzem: *Apple Siri*, *Google Assistant*, *Amazon Alexa* (*2. táblázat*).

	Apple Siri	**Google Assistant**	**Amazon Alexa**
Hangalapú felismerés	igen	igen	igen
Általános tudásbázis	igen	igen	igen
Zene lejátszás	igen	igen	igen
Mindennapi életben felmerülő feladatok (ételrendelés, foglalás készítés)	igen	igen	igen
Online vásárlás	igen	igen	igen
Navigálás	igen	igen	igen
Hírek feldolgozása	igen	igen	igen

2. táblázat Digitális személyi asszisztensek összehasonlítása [27][28][29][30][31]

A hírek feldolgozása terén azonban a korábban bemutatott chatbot megvalósítások talán abban járnak előrébb, hogy ott kategória begépelésére lehet szűkíteni a kört. Míg a hangalapú megoldások általában az előre megadott kategóriákat olvassák fel egy megadott médiacég választékából vagy éppen a kedvenc hír témájú podcastot indítja el. A *Google Assistant* például egész weboldalakat, valamint hír cikkeket képes felolvasni [27][28][29][30][31].

2.6. Szakirodalmi áttekintés összefoglalása

A virtuális asszisztensek és mesterséges intelligencián alapuló chatbotok elmúlt években tapasztalt térnyerése jelentősen megváltoztatta a számítógép és ember közötti interakció módját. Ez nagyrészben a *mesterséges intelligencia, IOT eszközök, gépi tanulás, természetes nyelvfeldolgozás és big data* területek fejlődésének köszönhető. A technológiai előrelépéseknek köszönhetően szinte bármilyen okoseszközön megtalálható a felhasználó igényeit leső virtuális asszisztens, aki különféle egyszerű feladatok megoldását, hang- vagy szöveges utasításokkal kommunikálva oldja meg.

A digitális asszisztensek elődjeinek tekinthetjük azokat a chatbotokat, akiknek elsődleges céljuk a felhasználóval való kommunikáció utánzása volt. Ezek a megvalósítások még a szövegben kerestek mintázatokat és előre programozott válaszokat adva működtek. Bár megtévesztő volt, működésük még kezdetleges, mesterséges, nem volt képes kontextusnak megfelelően követni egy-egy beszélgetés fonalát vagy teljesíteni egy felhasználó kérését.

Aztán a sokrétű fejlődésnek köszönhetően a beszélgető robotok egyre jobban képesek voltak az emberi kommunikációt utánozni. A neurális hálók terén tett előrelépés segítségével egy mondatból képesek vagyunk kinyerni a legfontosabb adatokat, a felhasználó által közölni kívánt szándékot, így meghatározható mi a kérése. Ezáltal a robot entitásokhoz tudja társítani a felhasználói szándékot, amivel a belső feldolgozó logika megvalósíthat egy külső jól felparaméterezett kérést egy külső adatbázissal vagy szolgáltatással.

A hírfeldolgozás a virtuális asszisztensekben megosztja a piacot, ugyanis a nagy cégóriások által nyújtott személyes digitális asszisztensek funkcionalitásai között, ha megtalálható a hírek személyes preferencia szerinti gyűjtése annak írásos összefoglalása, esetleg kategória szerinti keresése már nem minden esetben állítható tisztán. Azonban a chatbot megvalósításoknál már nem ritka hasonló alkotás, viszont ezek túlnyomórészt webes alkalmazásként érhetőek el.

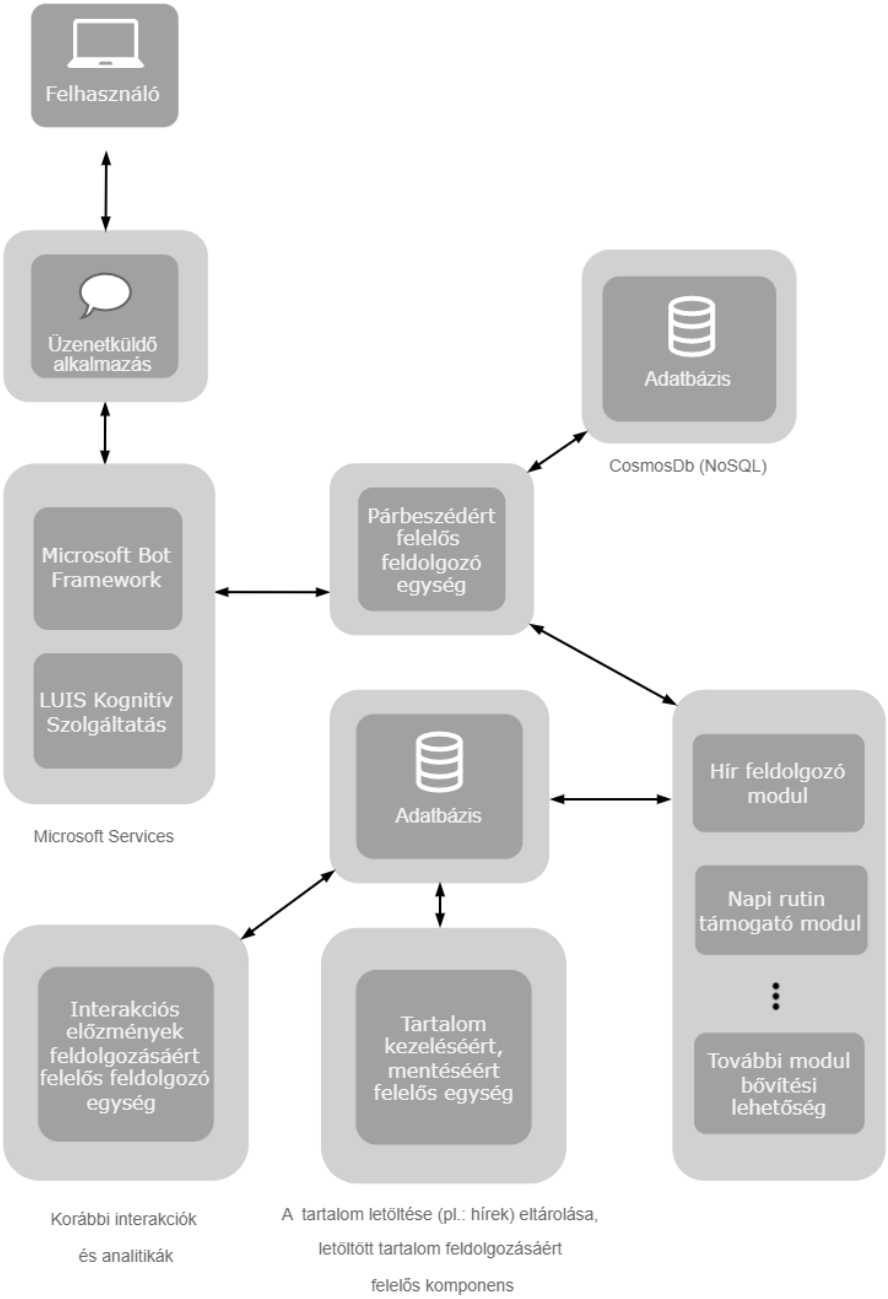
Szakdolgozatom célja egy olyan rendszer felépítése, ami - a piackutatás alapján jelenleg hiánynak mondható - egy olyan asztali alkalmazás megalkotása, amely a felhasználó preferenciáit figyelembe véve képes a hírek feldolgozására, illetve egyéb modulokkal kiegészítve lehet a funkcionalitások körét bővíteni úgy, hogy az alap hírfeldolgozó modul akár webalkalmazásba, ismert csevegőalkalmazásokba is beépíthető legyen.

3. TERVEZÉS

A fejezetben egy előzetes rendszerterv kerül bemutatásra, amelyben előre meghatározásra kerül a használni kívánt technológia és a megvalósítandó funkcionalitások köre.

3.1. Rendszerterv

A rendszer kezdeti felépítése a *3.1.1. ábrán* látható.



3.1.1. ábra Rendszerterv

Az alkalmazás két fő részből áll az asztali *WPF* alkalmazás, ami a Microsoft Bot Frameworkot használva képes úgynevezett REST API hívásokkal kommunikálni a Bot Portalon futó saját implementációjú robottal. A Microsoft Bot Framework remek lehetőséget nyújt továbbá a felhőalapú csevegőalkalmazások implementálására, kognitív szolgáltatásokat is nyújt, amivel a robot megérti a természetes nyelvet és képes a szándékot az üzenetből kinyerni. Továbbá a robot tudásbázisaként egy saját hírfeldolgozó komponens is részét képezi a feladatnak, amely a híreket letölti, kategorizálja egy külső komponens segítségével a lényegi információt összefoglalja és eltárolja. A hírfeldolgozó komponens mellett a napi rutin támogatása is része a programnak, amely a teendők felvételét és lekérdezését, egy tetszőleges weboldal megnyitását, fejlesztői környezet beállítását és egy tetszőleges weboldal elérhetőségének ellenőrzését tartalmazza. Az adatbázis a Azure SQL adatbázis segítségével valósul meg. Valamint egy saját interakciók feldolgozásáért felelős modullal is rendelkezik az alkalmazás, amely a felhasználó szokásait képes a korábbi preferenciák és a válaszokon alapulva meghatározni (*3.1.1. ábra*).

A választott eszköz a megvalósításra a *Microsoft Bot Framework,* ennek oka a *Microsoft* által nyújtott komplex szolgáltatási csomag, amely magában foglal minden olyan külső a feladatban szereplő egyéb szolgáltatást (pl.: *Bing News API, Azure Cognitive Services, LUIS* stb.), amelyek egy helyen elérhetőek. Továbbá fontos volt a megvalósításánál figyelembe venni, hogy lehetőleg a *C#* nyelv alkalmazásával valósuljon meg a feladat.

4. MEGVALÓSÍTÁS

A fejezet tartalmazza a megvalósítandó rendszer részletes ismertetését, a felhasznált technológia és külső szolgáltatások felhasználásának jellemzését, majd az elkészült alkalmazás funkcionalitásainak áttekintését és képernyőképekkel alátámasztását.

4.1. A rendszer ismertetése

A rendszer két fő részből áll, ezek a natív felhőtechnológiát használó webalkalmazás (chatbot) és az ezzel kommunikáló asztali alkalmazás, amely lehetőséget teremt a felhasználó számára, hogy interakcióba léphessen a robottal. A program moduláris felépítést követ, ez lehetővé teszi, hogy a jövőben újabb funkcionalitásokkal egyszerűen bővíthető legyen. Ez azt jelenti, hogy egy-egy funkció saját dinamikus csatolású könyvtárként lett megalkotva (*dll – dynamic link library*).

A beszélgető robotalkalmazás a *Microsoft Bot Framework SDK* a természetes kommunikáció támogatására használt keretrendszer felhasználásával készült el a, míg a kliens alkalmazás a *WPF* (*Windows Presentation Foundation*) grafikus felhasználói felületek tervezéséhez használt keretrendszer segítségével, mindkét alkalmazás *C#* nyelven valósult meg. A *.NetCore 3.1* verzió került felhasználásra minden projekt esetében, ugyanis a *Microsoft Bot Framework SDK* jelenleg nem támogat ennél frissebb verziót, továbbá az egyes projektek közötti függőségek miatt az eltérő verziók használata így nem lehetséges.

Az alkalmazás az *Azure* szolgáltatásait használva került telepítésre egy *Azure App Service* erőforrás felhasználásával. Erre azért van szükség, mert saját fejlesztésű kliens applikáció csak akkor képes kommunikálni a *Microsoft Bot Framework SDK* segítségével készült robotalkalmazással, ha az egy felhőszolgáltató virtuális gépén kerül hosztolásra. Ezt a kommunikációt a *Microsoft Direct Line Channel* nevű technológiája teszi lehetővé, ami a webes erőforráshoz egy *REST API* végpontot biztosít. Ezt felhasználva létrejöhet a kapcsolat a kliens alkalmazás és a chatbot között. Továbbá a gördülékeny telepítési folyamat érdekében a forráskód a *GitHub* verziókezelő rendszerben publikálásra került, ez hozzákapcsolódik a virtuális géphez az *Azure App Service*-ben, ezt felhasználva minden egyes változás a verziókezelő rendszerben kivált egy frissítési kérést az alkalmazásban, amely ennek hatására automatikusan telepíti a friss változtatásokat és nincs szükség beavatkozásra a felhasználó részéről, ezt a műveletet folyamatos üzembehelyezésnek (*continuous deployment*) nevezik.

Az *Azure App Service* szolgáltatás keretein belül számos opció van, amelyek különböző árkategóriákat jelentenek. A költségek minimalizálása érdekében a kis kapacitású virtuális gép választása indokolt, ennek következtében néhány limitáció megjelenik a rendszer fejlesztése során. A virtuális gép jellemzői:

- CPU-használat: 60 perc / nap, 5 percen belül nem lehet 3 percnél több,

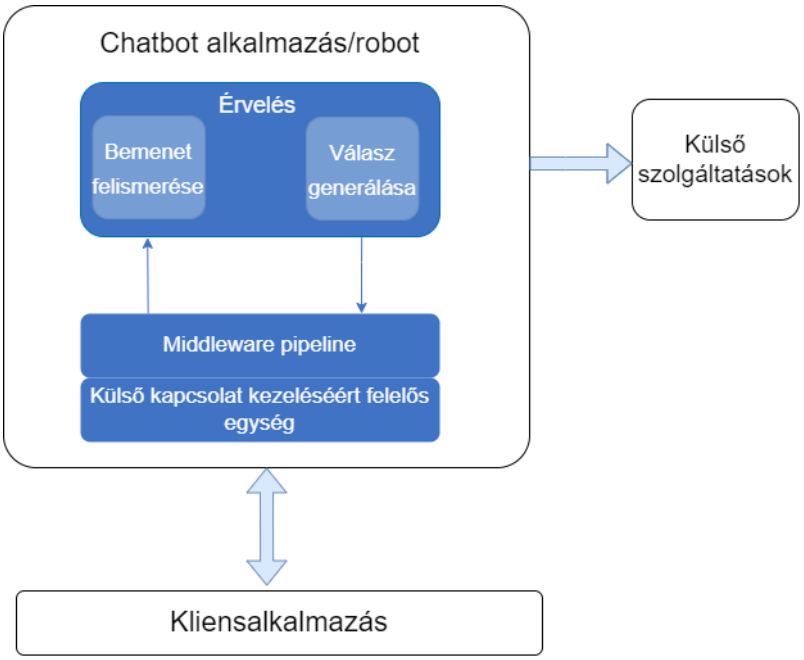
- Memória használat: 1024 MB / 1 óra,
- 165 MB sávszélesség [35].

4.2. A chatbot bemutatása

A fejezet bemutatja a felhasznált technológiákat, általános paradigmákat, amelyek elengedhetetlenek a chatbot fejlesztése során.

4.2.1. Microsoft Bot Framework SDK

A *Microsoft Bot Framework SDK* az intelligens robotok létrehozásához, teszteléséhez, kezeléséhez és üzembe helyezéséhez biztosít eszközöket a fejlesztők számára. A keretrendszer magába foglal olyan szolgáltatásokat is, amelyekkel az intelligens párbeszéd imitálható, ilyen például az AI szolgáltatás, amely természetes nyelv megismerésére vagy akár beszéd használatára is alkalmas. A keretrendszer integrált eszközkészlettel rendelkezik, amely a teljes fejlesztési időszakban hasznos eszközökkel segíti a fejlesztők munkáját, ezek a területek a tervezés, implementáció, tesztelés, publikálás, különböző csatornák csatlakoztatása, eredmények kiértékelése [36].



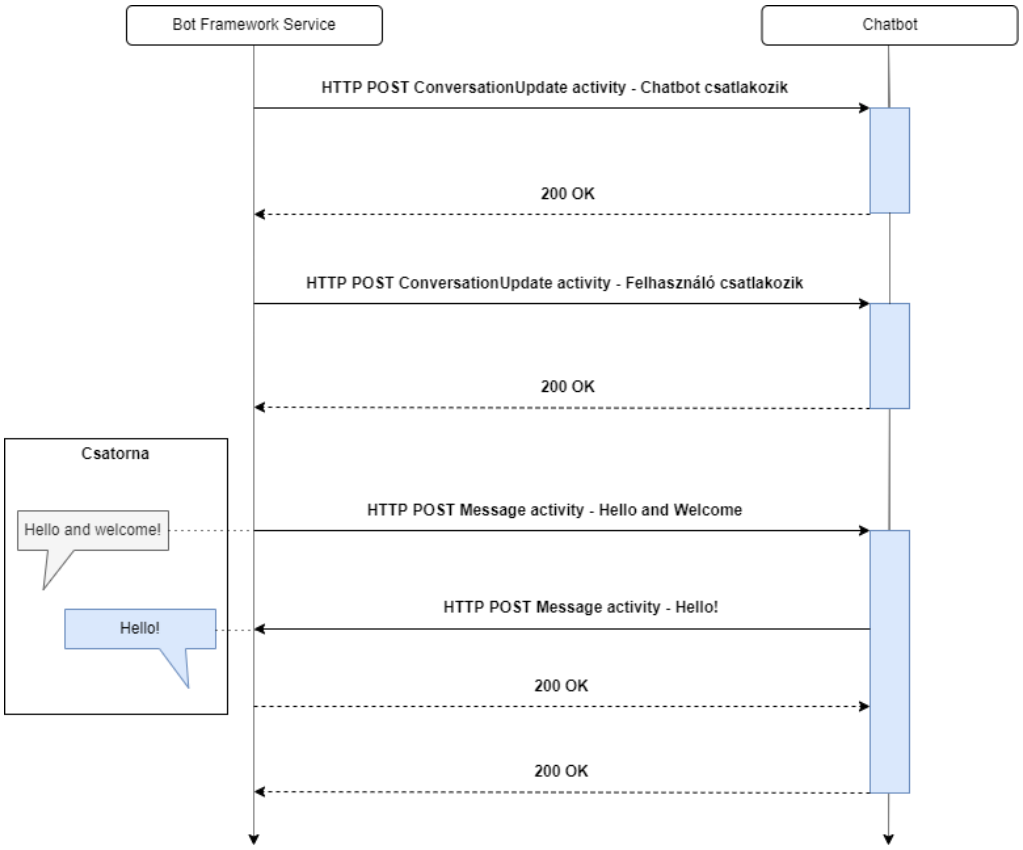
4.2.1. ábra Microsoft Bot Framework működése [37]

A *4.2.1. ábra* akár egyszerűsített rendszertervként is értelmezhető, azonban a *Bot Framework SDK* nyújtotta lehetőségeket is jól szemlélteti. A chatbot a bemenetet képes felismerni és a megfelelő belső logikát tartalmazó egységhez fordulva választ generálni. Azt a közeget, amelyben az információ a kliens és chatbot alkalmazás között áramlik, csatornának nevezik. A *Bot Framework Service* a korábban említett *Direct Line Channel* mellett számos más opciót is nyújt, többek között a *Microsoft Teams, Facebook* és *Slack,*

amellyel üzenetek és események küldése is támogatott, így a robothoz való csatlakozás megoldható saját kliensalkalmazás fejlesztése nélkül is. Az *Azure Bot Service* további szolgáltatások bevonásával kiterjeszti a rendelkezésre álló lehetőségeket, mint például az *Azure Cognitive Services*, amellyel intelligens chatbot alkalmazások építhetők, valamint az *Azure Storage*, ami felhőalapú tárhelyszolgáltatást biztosít. A chatbot manuális teszteléséhez is adott a keretrendszeren belül több lehetőség is. A lokálisan működő chatbotok esetében az *Bot Framework Emulator* nevű program nyújthat segítséget. Ez a program kifejezetten hasznos a robot webes környezetbe telepítése előtt, ugyanis azután már a virtuális gépen futva az *Emulator* program nem tud támogatást adni a hibák felderítésére. A hosztolt alkalmazás esetén az *Azure Application Insights* nevű erőforrás nyújt megoldást arra, hogy a használat közben kiváltott kivételeket naplózza és részletes leírást adjon arról, miért következhetett be a hiba. Például az alkalmazás felhőbe telepítése után az elérési útvonalak megváltoztak, mert egészen más mapparendszerben kerültek kihelyezésre az *Azure* tárhelyére, mint ahogyan azt a lokális mapparendszer korábban definiálta. Ennek megoldásában nagy segítségre volt az *Azure Application Insights,* hiszen a kiváltott kivétel szövege alapján egyértelműen beazonosíthatóvá válik a hiba.

4.2.2. Általános chatbot alkalmazásstruktúra

A csevegő robot és a kliensalkalmazás közötti csatornán úgynevezett *activity* objektumok tartalmazzák az egyik féltől a másiknak küldött információt. A *Bot Framework Service* által biztosított *activity* objektum a csatorna típusától függően eltérő adatot tartalmazhat. Kijelenthető, hogy az alkalmazások többségében az *activity* objektumok eseményvezértelt módon jutnak el a kliensalkalmazástól a bot applikációig.



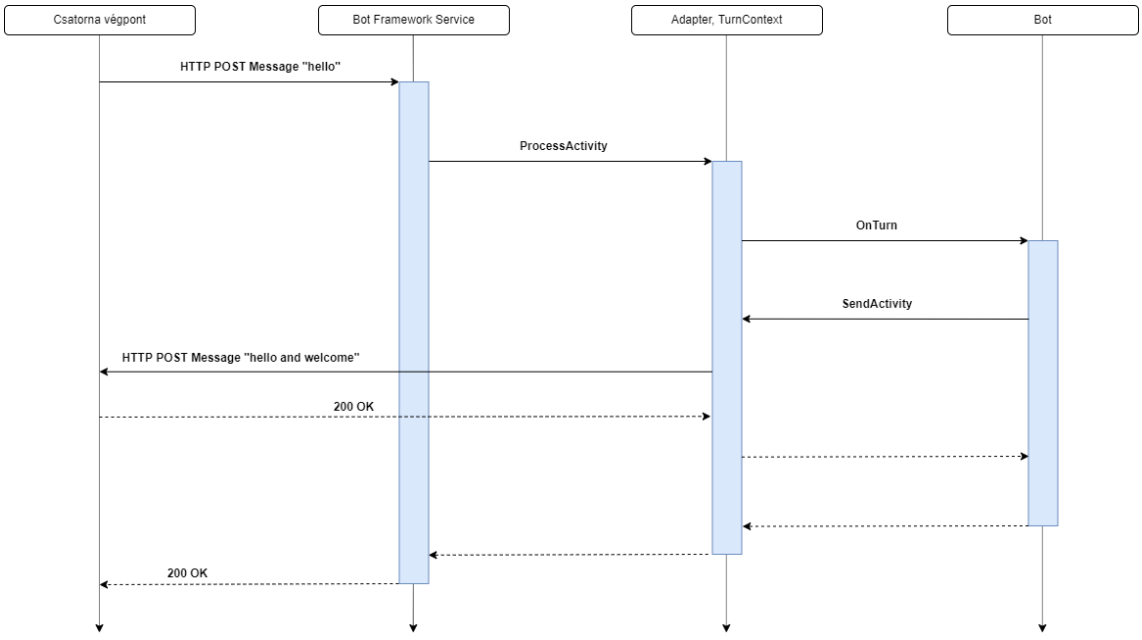
4.2.2. ábra Bot Framework Service párbeszédmodell [37]

Az *OnMembersAddedActivity* esemény például a felek csatlakozásakor használatos a keretrendszer által. Alapvető része a párbeszéd kialakításának, ugyanis a rendszerben ehhez, lehet kötni a felek jelenlétét, amely egyértelmű belépési pontot jelent. Az *activity* rendelkezik egy *MembersAdded* tulajdonsággal, amely alapján meghatározható ki csatlakozott a beszélgetéshez. Fontos megemlíteni azonban, a *Direct Line Channel* sajátosságát, hogy csak akkor küldi el a *4.2.2.* ábrán is található felhasználó csatlakozását jelentő *ConversationUpdate activity* objektumot, miután a felhasználó már elküldte első üzenetét, amely azt jelenti, hogy a robot csak akkor juttatja el saját üdvözlő üzenetét, ha a felhasználó már előtte kezdeményezte volna a párbeszédet. Ezért a BotFramework Service által biztosított egyedi esemény kiváltása jelentett megoldást. A program indulásakor a kliens alkalmazás generál egy eseményt egy tetszőlegesen választott azonosítóval, amelyre a chatbot az *OnEventActivity* metódusában fel van iratkozva és a

szükséges választ generálja, jelen esetben a felhasználó üdvözlésére volt ilyen módon szükség. Ez azért kiemelendő, mert nagy jelentősége van a párbeszéd modellezésekor annak, hogy a felhasználót a robot üdvözölje először és ne a felhasználó első üzenete okozza az egész beszélgetés folyamatának beindítását.

Az interakciók során tehát *activity* objektumok kerülnek átküldésre a felek között. A chatbot és a felhasználó által küldött üzenet-üzenet párokat fordulónak nevezik, vagy másnéven beszédlépésnek (angolul *turn*). Egy fordulóban, ami lehet egy kérdés-válasz, vagy akár egymás kölcsönös üdvözlése is, a felhasználó beérkező *activity* objektuma és a chatbot kimenő *activity* objektuma vesz részt. Például, ha a felhasználó kérésére a robotnak további kérdést kell feltennie a beszélgetés folytatásához, akkor a beszédforduló ezzel a kérdéssel ér véget. Ebben az esetben a felhasználó válasza már egy következő fordulóban érkezik vissza a robothoz [40].

Az *SDK* használatával készült csevegőbotok kulcsfontosságú alkotóeleme a bot adapter, amely a HTTP forgalmat közvetlenül kezeli. Az *adapter* részére kerül elküldésre a *HTTP request*, amelynek body részét *activity* objektummá konvertálja. Az adott beszédfordulóban az adapter hozza létre a beszélgetés kontextusát, amelyben az aktuális információk kerülnek eltárolásra az egyes fordulók között, mint például a küldő és fogadó adatai, a csatorna adatai, az *activity* feldolgozásához szükséges egyéb adatok. Továbbá rendelkezik egy *middleware pipeline*-al is, amelyben az *activity* feldolgozása egészíthető ki még mielőtt a *bot turn handler* metódusa feldolgozná, majd a bot *turn handler* feldolgozás után is lehetőség nyílik az *activity* feldolgozására a *middleware* segítségével (*4.2.3. ábra*) [40].



4.2.3. ábra Activity objektum feldolgozása [40]

Egy példán keresztül bemutatva az alábbi folyamatot járja be az *activity* objektum a kliensalkalmazás és a robot között (a példában a robot „*hello and welcome*” üzenettel válaszol a felhasználó „*hello*” üzenetére):

1. A *Direct Line Channel* végpontjára beérkezik egy üzenet a kliensalkamazástól, a webszerver fogadja a *HTTP POST request*-et. Az activity *JSON* payloadként érkezik a *HTTP POST request body* részében, ahol deszerializálás után a további feldolgozás a *ProcessActivity* metódusban folytatódik, ahova paraméterként átadásra kerül.
2. Az *adapter* a beszédlépéshez készít egy új kontextust és a *middleware*-t hívja meg. A *middleware* hívás után az adapter a bot forduló kezelőjét hívja (*turn handler*) és az *activity* objektumot adja át.
3. A bot forduló kezelője megkapja az *activity* objektumot és elküldi válaszát a végpontnak.

A *bot* objektum tartalmazza a beszélgetés érvelés (angolul *reasoning*) részegységét (*4.2.1. ábra*), amely az *adapter*től érkező *activity* objektumokat fogadja és elvégzi a kívánt üzleti logikát. Az említett logika kezelésére a keretrendszer a következő paradigmákat vezeti be:

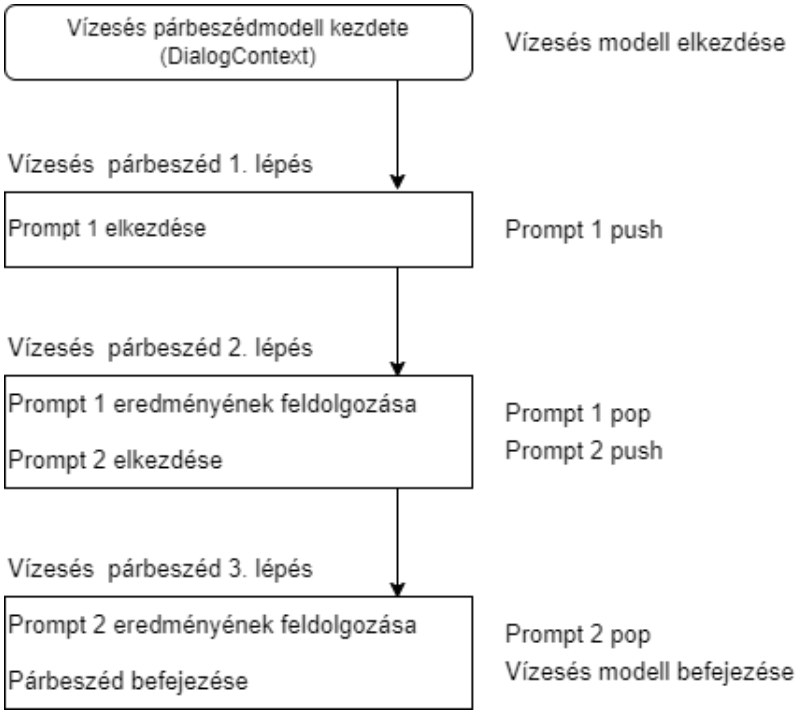
- *Activity* kezelők: egy esemény alapú modell, amelyben az összes létező esemény típusra saját kezelő metódus implementáció definiálható,
- Párbeszéd kezelők: előre definiált párbeszédmodelleket nyújt a keretrendszer, amelyben az állapotkezelés is támogatott,
- Egyedi megvalósítás: természetesen lehetőség nyílik a saját implementációra is, amennyiben az igényt nem fedik le a keretrendszer nyújtotta lehetőségek [37].

A dolgozat megvalósítása során a feladat az alapvető interakciók támogatása volt, amelyhez elegendő az *SDK* által nyújtott előre definiált *activity* kezelők és párbeszédmodellek használata. A feladat megoldásakor az *ActivityHandler* beépített osztály került felhasználásra, amelynek használata során a saját bot osztály őseként kell definiálni az *ActivityHandler* osztály, így az örökölt metódusok között az összes a keretrendszer által támogatott eseménykezelő metódus elérhető. Az elérhető metódusok közül a korábban említett *OnEventActitvity* mellett az *OnMessageActivity* és *OnTurn* metódusok kerültek felhasználásra, amelyek a továbbiakban kifejtésre kerülnek. A párbeszédek kezelésénél pedig az úgynevezett komponens vagy összetevő (*component dialog*) párbeszéd modell került alkalmazásra [37].

Az összetevő párbeszéd modell egy olyan modellezési forma, amely arra ad megoldást, hogy egy párbeszéden belül meg lehessen hívni akár több másik párbeszédet is, így lehetőséget nyújtva arra, hogy kiterjeszthető legyen olyan egyedi használati esetre is, ahol többszintű hierarchiában szerepelhet egy vízesésszerű párbeszédmodell (*waterfall dialog*) és egy egyszerű kérdés-válasz (*prompt dialog*) társalgási modell is. A vízesés

párbeszéd modell szintén az *SDK* beépített funkcionalitása, ez a modell egy előre rögzített szekvenciát definiál, amelyen végig vezeti a felhasználót, miközben a megfelelő információkat begyűjti és az alapján végzi el a kért műveletet. Az információk begyűjtéséhez az egyszerű kérdés-válasz (*prompt dialog*) társalgási modellt használja [38].

A vízesés párbeszéd modell célja, hogy végig vezesse a felhasználót lépésről lépésre az adott feladat elvégzéséhez szükséges lépéseken. Az egyes lépések aszinkron metódusként vannak definiálva és minden lépés kontextusa a metódus paraméterében kerül átadásra (*WaterFallStepContext*). A chatbot minden egyes lépésnél jellemzően valamilyen adatot kér be a felhasználótól, amelyet feldolgoz és a következő lépéssel folytatja. Két beszédlépés között azonban szükség lehet az előző körben begyűjtött információra, ehhez nyújt támogatást a kontextus változó, ugyanis ennek a megfelelő paraméterében eltárolásra kerül az előző forduló eredménye. A *4.2.4. ábra* szemlélteti a vízesés modell lépéseit, valamint az utasításokat, amit egy verem adatszerkezet formájában lehet elképzelni [38].



4.2.4. ábra Vízesés párbeszéd modell [38]

4.2.3. Robot állapotának kezelése

A webalkalmazásokra általánosan jellemző, hogy nem tartanak fent állapotot, és mivel a chatbot is webalkalmazás, így itt is hasonló megoldásokat kell alkalmazni. Az *SDK* azonban szolgáltat beépített állapot fenntartására használható eszközöket. Természetesen nem minden chatbot esetében van szükség olyan tár építésére, amely az alkalmazás egyszeri életciklusán átívelő időtartamban is perzisztens, amennyiben a robotnak nincs

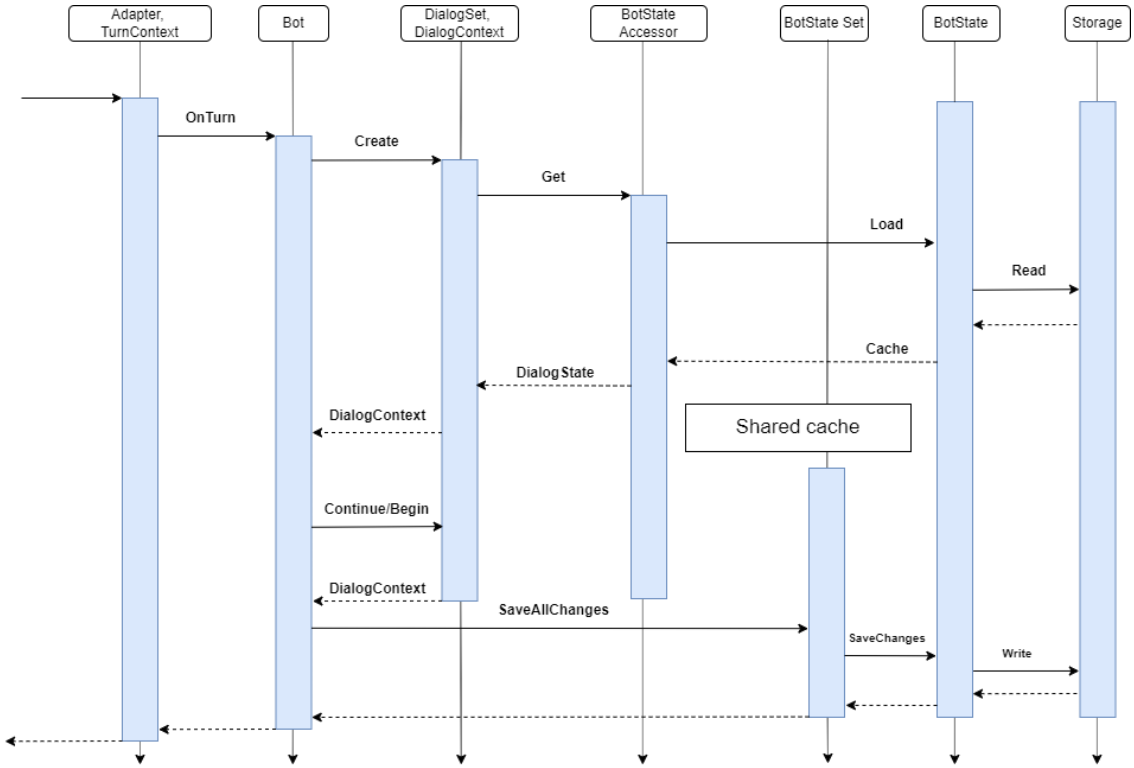
olyan funkciója, amely erre épít kihagyható a fejlesztési folyamatból. A dolgozat témája sem feltétlenül indokolja a használatát, a kísérletezés volt a fő cél ennek bevezetésekor. A megvalósult funkció keretében a beszélgető alkalmazás a felhasználó nevét kéri be, de csak abban az esetben, ha ez még nem történt meg korábban. Amennyiben már rendelkezik az információval a felhasználót a nevével együtt képes üdvözölni.

A funkció implementálásához az *Azure*-ben saját erőforrásként létrehozott *CosmosDb* példány létrehozása volt az első lépés. Az *Azure CosmosDb* egy partícionált tárhely, amely egy *CosmosDb NoSQL* adatbázishoz csatlakozik (*4.2.5.* ábra). Az *SDK* integrált állapotkezelése 3 különböző területet különböztet meg, amelynek segítségével adattárolás valósítható meg, ezek a területek:

- Felhasználó állapota,
- Párbeszéd állapota,
- Privát párbeszéddel kapcsolatos állapot [39].

A fenti felsorolásban szereplő állapotok közül a felhasználó állapotának felülírására volt szükség a feladat megvalósításához. A felülírás egy saját objektummal történik, amely az adott célnak megfelelő struktúrát vehet fel, természetesen a *NoSQL* adatbázisban az adatok majd ezzel a felépítéssel fognak rendelkezni. A felhasználó állapota minden beszédfordulóban elérhető, így annak tartalma bármikor kiolvasható. Az állapot értékek írása és olvasása az állapot-tulajdonságok (*state properties*) segítségével működik, ami annyit tesz, hogy a megadott értékek kulcs-érték párokként képződnek le a háttérben, így a korábban definiált objektum struktúra lesz a felhasználó állapot kulcsának az értéke [39].

Az állapot tulajdonság módosítók (*state property accessors*) abban állnak rendelkezésre, hogy az állapot írható és olvasható legyen. Az adott módosító a bot létrejöttekor kerül példányosításra egy megadott névvel, ezután már a tulajdonság segítségével elérhető és módosítható a hozzákapcsolt állapot (*4.2.5. ábra*) [39].



4.2.5. ábra Bot állapotának kezelése [39]

4.2.4. Adattárolás

Az alkalmazás adattárolása egy felhőalapú adatbázis segítségével valósult meg, a választás azért esett az *Azure SQL Database* szolgáltatóra, mert a feladat egyéb részeinél is a Microsoft technológiák használtak, továbbá az *Azure For Students* promóció keretében ajánlott kreditek felhasználásával a költségek minimalizálása zökkenőmentesen valósulhatott meg. Természetesen a legfőbb szempont az volt, hogy a webalkalmazás elérhesse az adatbázist, így a lokális adatbázis már nem jöhet szóba. Ennek előnye akkor is megmutatkozott, amikor a virtuális géppel kapcsolatos limitációk (*lásd 4.fejezet*) nem tették lehetővé a webalkalmazás és a kliensalkalmazás közötti kommunikációt megnövekedett adatmennyiség esetén. Ennek legfőbb indoka, hogy ebben a díjkategóriában a maximális operatív memória használat 1024 MB / 1 óra mennyiségben maximalizált. Azonban egy adatbázis lekérés következtében elhasznált memóriamennyiség jelentősen meghaladja ezt a limitet így nem hagyva lehetőséget az alkalmazás számára, hogy tovább fusson. Erre a problémára természetesen megoldás lehetne, hogy a virtuális gép egy magasabb szolgáltatáscsomagba kerüljön hosztolásra, azonban ez nagy mértékben megnövelné a költségkeretet, ami nem fér bele az *Azure For Students* által meghatározott 100USD/év maximális kiadásba. Emiatt az alkalmazásban meg kellett törni azt az alapvetést, hogy a webalkalmazás szolgáltat minden üzleti logikát és a kliensalkalmazás csak a megjelenítésért felel. Normális körülmények között ez természetesen nem ezen módszer szerint lenne kialakítva, viszont itt nem állt

rendelkezésre más alternatíva, így a hírek megjelenítésekor a kliensalkalmazás végzi ezt el, ez a továbbiakban bővebben kifejtésre kerül.

Az alkalmazásban a hírek, a felhasználó teendői, valamint a preferenciák tárolása valósul meg. A hírek tárolásával kapcsolatban részletesebben a 4.2.6. fejezet foglalkozik. A teendők és preferenciák tárolása az alapvető adatigényeket fedi le. A teendők létrehozása a felhasználó által kért dialógusban a bot által megadott rögzített lépések mentén zajlik, erről bővebben a 4.2.4. fejezet szól. Ehhez a funkcionalitáshoz az eltárolt adatok köre az adott felhasznált azonosítója, a teendő neve és a teendő határidejében merül ki. A felhasználó preferenciáinak tárolására, azért van szükség, mert azalkalmazásnak a feladatkiírás szerint tudnia kell a felhasználó szokásait megismerni. Ehhez az alkalmazás azon pontjain, ahol a felhasználó valamilyen választást végez azzal kapcsolatban, hogy mit kér az alkalmazástól - például milyen kategóriában szeretne híreket látni - történik egy mentés a választott kategóriával kapcsolatban. Tehát közvetlenül nem módosítható a tábla a felhasználó által. Ezen funkcionalitás tervezésekor szükség volt arra, hogy az adattárolás rugalmasan tudjon zajlani, a rugalmasság alatt itt az eltárolt adatok sokszínűsége értendő. Emiatt minden területnél létrehozásra került egy modell, ami az eltárolt adatokat az adott igényeknek megfelelően tartalmazza és azt a tábla „*Payload*” tulajdonságába szerializálva szövegként tárolja. Emellett egy enumeráció került létrehozásra, amely az adott területet jelenti, tehát például a hírek letöltésénél a megfelelő enumeráció típus és a választott kategória nevére van szükség, míg például a böngészési szokások rögzítésénél a választott weboldal címe kerül eltárolásra szintén a megfelelő enumeráció típussal. Ennek előnye, hogy egy esetleges későbbi bővítés esetén nincs szükség az adatbázis séma bővítésére, azonban hátrányként megjelenik a szöveges tartalom típusos objektummá konvertálása, amely egy jelentősen erőforrásigényes művelet lehet nagyobb adatmennyiség esetén.

4.2.5. Moduláris felépítés támogatása

Ahogyan az korábban már említésre került az alkalmazás moduláris szemlélet mentén került kialakításra. Ennek megvalósítása a .NET alapú alkalmazásokban az *Assembly* osztály segítségével valósulhat meg. Az *assembly* tulajdonképpen az alapvető építőeleme egy .NET alapon nyugvó applikációnak a telepítés, verziókezelés és újrafelhasználás szempontjából. Az *assembly*ben meghatározott típusok és erőforrások alkotják azt funkcionális és logikai egységet, amely az adott feladat megvalósítása során előáll. Kiterjesztésük szempontjából lehetnek futtatható (*.exe*) vagy dinamikus csatolású könyvtár (*.dll*) fájlok [40]. A moduláris felépítéshez az adott funkcionalitásokat az előbb említett *.dll* kiterjesztésű fájlként kellett megvalósítani. Így minden egyes adott specifikus területhez kötött műveletvégzés előtt meg kellett vizsgálni, hogy az adott könyvtár létezik-e és ha igen, akkor a megfelelő könyvtár példányosítása után a szükséges metódust meghívni. Ezen a területen jelentkezett probléma az alkalmazás webes környezetbe telepítéskor, ugyanis a projekt mappa struktúrája telepítés után nem a lokális

esetben meghatározott mappahierarchia. Annak ellenére, hogy a megadott *.dll* könyvtár elérése relatív hivatkozási móddal került megadásra az eredeti elképzelés szerint, az *Azure*ben a telepített alkalmazások a „*C:/home/site/wwwroot/*” mappa alá kerülnek. Tehát minden dinamikusan létrehozott könyvtár az adott gyökérkönyvtár alá lesz elhelyezve, így azok hivatkozása is ezzel a gyökérmappával kezdődően kell, hogy történjen.

4.2.6. Hírfeldolgozó egység

A felhasználó napi tájékozódásának segítése a hírek csoportosítása, tárolása, megjelenítése és a csevegőalkalmazásba integrálásával valósul meg. Ahhoz, hogy a lehető legfrissebb hírek is a felhasználóhoz eljuthassanak egy saját RSS-hez hasonló működésű komponens készítése volt szükséges. Ennek alapja a *Microsoft Bing News API*, amellyel a hírek a megfelelő struktúra szerint letöltésre kerülnek és az alkalmazás a megfelelő formátumba konvertálása után ezt a korábban említett *Azure SQL* adatbázisba eltárolja. A feladatban kiemelt szerepet kapott, hogy olyan módon kell az híreket eltárolni, hogy ahhoz kategória információ is legyen, ezért is esett a *Bing API*-ra a választás, mert a híreket egy előre meghatározott, egyszerű lista szerint osztályozza, amelyben a következő értékek lehetnek: *üzlet, szórakozás, egészség, politika, termékek, tudomány és technológia, sport, Egyesült Államok, világ.* Az adatbázis feltöltésénél arra kellett megoldást találni, hogy a megfelelő kategória is el legyen tárolva, ugyanis a *Bing API* azt az adatot nem foglalja bele az általa szolgáltatott eredménybe. Ezért azt a lehetőséget kellett kihasználni, amely csak az adott kategóriához tölt le híreket, viszont ezt az összes kategóriánál meg kell tenni. Ezáltal az összes hír a megfelelő kategória adatával együtt lesz a rendszerben.

Továbbá a hírek feldolgozásakor még a feladat része volt az is, hogy ezeknek összefoglalása is történjen meg. Ehhez szintén külső, az *Azure Cognitive Services* szolgáltatás került felhasználásra. Ez a szolgáltatás képes arra, hogy a megadott szövegből kinyerje azokat a mondatokat, amelyek kollektívan az egészet tekintve a legfontosabb és legrelevánsabb információkat tartalmazza. Az alkalmazásba a REST API segítségével került beépítésre, a mentési folyamat részeként minden egyes letöltött cikknél az összefoglalt tartalom már az elmentett adat része. A felhasznált komponensnek kívülről megadható, hogy maximum hány mondatban van szükség a szöveg tartalmára, itt 1 mondat került beállításra. Erre az adatra majd a megjelenítéskor lesz szükség, itt ugyanis csak a rövid tartalom jelenik meg a cikk címével együtt. Az eltárolt adatok köre az alábbiakban merül ki:

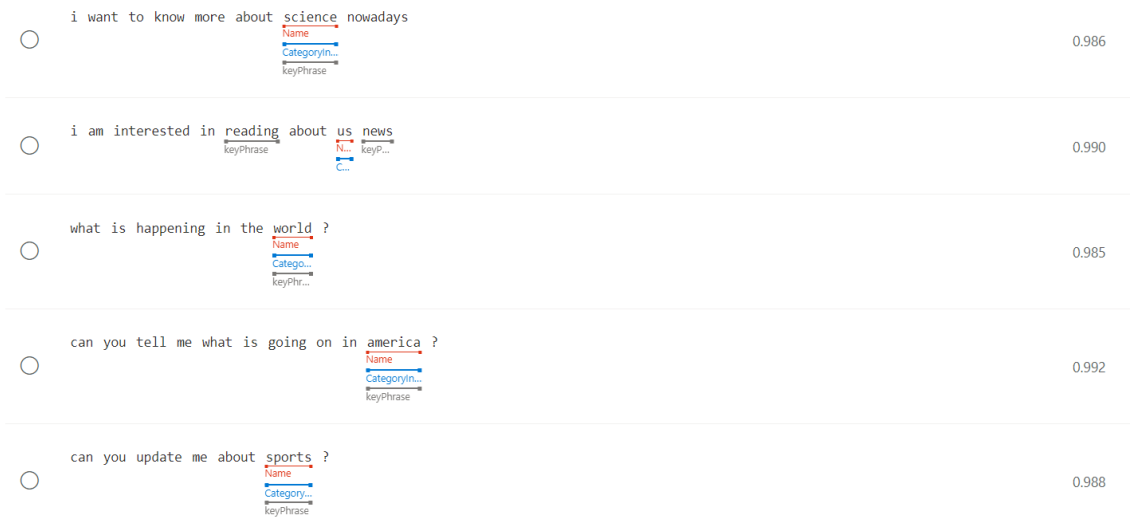
- cikk címe,
- cikk url,
- kategória neve,
- cikk tartalma,
- cikk összefoglalása,

- kép url,
- kép mérete (szélesség, magasság),
- kép származási helyének adatai (név, url, licensz),
- publikálás dátuma.

4.2.7. LUIS

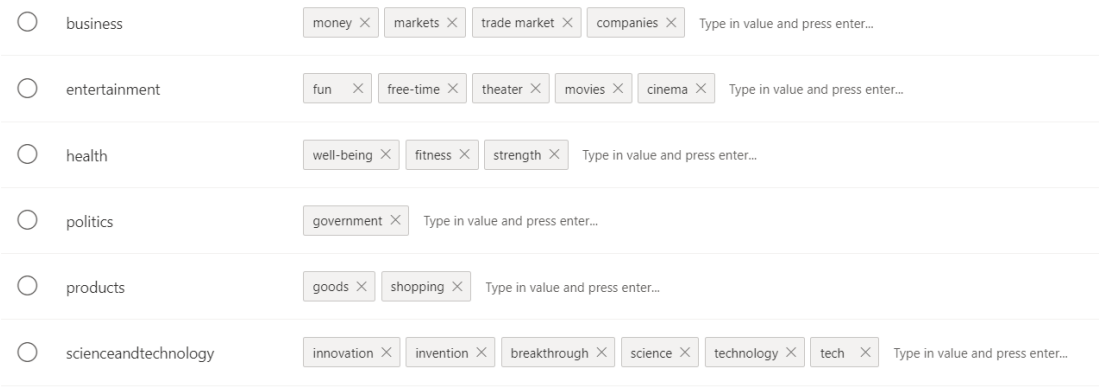
A *LUIS* betűszó a *Language Understanding* angol szavak első két kezdőbetűiből származik, melynek jelentése a nyelv megértéseként fordítható. Ez egy felhőalapú beszélgetési AI szolgáltatás, amelynek segítségével a felhasználó természetes nyelvi szövegéből általános jelentés előrejelzés és releváns, részletes információkinyerés valósítható meg [42]. A szakdolgozatban a felhasználása a hírek modulban került felhasználásra, ahol a vízesés társalgási modellel együtt a felhasználó üzenetéből a releváns információ kinyerése a *LUIS API* felhasználásával történik. Ennek beépítése kiterjeszti a csevegés élményét és kevésbé tűnik a robot használata egy menürendszernek. Ebben a rendszerben a mély tanulás algoritmus mellőzésre került, itt az került kipróbálásra, hogy a felhasználó által megadott bemenetre a hír kategóriájához hogyan támogatható a *LUIS* segítségével a rokon értelmű szavak megadása, ezzel támogatva azt, hogy a felhasználó saját szavaival is megadhassa azt és ne feltétlen a megadott listából kelljen választania.

Ehhez saját LUIS alkalmazás létrehozására van szükség a luis.ai (*utoljára megtekintve: 2022.05.12.*) oldalon, ahol az *Intents,* azaz *szándékok* felvételével indul az alkalmazás fejlesztése. Ezt egy grafikus felhasználói felületen lehet elvégezni, nem kell hozzá alkalmazás kódot írni. Tehát a szándék a LUIS alkalmazásban azt a szándékot jelöli, amelyet a felhasználó végre szeretne hajtani. Jelen esetben a hírek letöltése, majd meg kell határozni az entitásokat. Tulajdonképpen ehhez *LUIS*-nak szükséges megadni olyan mondatokat, amelyek előfordulhatnak az adott szituációban és bejelölni azokat a szavakat, amelyek ebben a helyzetben érdekesek lehetnek, ezeket entitásnak nevezik (*4.2.6. ábra*).



4.2.6. ábra Szándékhoz tartozó példamondatok

Az entitások ez esetben a korábban felsorolt lehetséges kategóriák, amelyek egy listában kerülnek eltárolásra és az adott elemhez megadható szinonimák is itt adhatók meg (*4.2.7. ábra*).



4.2.7. ábra LUIS applikáció entitások és szinonimáik

Ezek után a *LUIS* alkalmazás tanítása következik, amely a megadott példák alapján megtanulja a mintázatokat és ezután már képes felismerni az erre illeszkedő bemeneteket. Ehhez az alkalmazáshoz *REST API* segítségével kapcsolódik az alkalmazás és küldi el a felhasználó bemenetét, amelyre az alkalmazás egy előrejelzés szerint megadja, hogy mekkora valószínűséggel lehet az adott bemenetre az általa választott kimenet jól illeszkedő.

4.3. Az asztali alkalmazás bemutatása

Azt asztali alkalmazás egy mediátor a felhasználó és a webalkalmazás között, amelynek fő feladata a robot üzeneteinek fogadása és a felhasználó üzeneteinek a robot felé továbbítása. Az üzenetek letöltésére a *polling* módszert alkalmazza a robot, ami tulajdonképpen azt jelenti, hogy egy végtelen ciklusban folyamatosan lekéri az *activity* objektumokat és *watermark* változó segítségével meg tudja határozni melyik *activity* az, amit a kliens applikáció még nem látott, ezen kívül egyszerűen meghatározható az *activity* objektumban, hogy ki az adott üzenet feladója, tehát már csak a megfelelő megjelenítést kell elvégezni. Az *activity* objektumok lekérése *HTTP GET request* segítségével valósul meg, amely az *Azure*ben regisztrált virtuális géphez (*bot példány*) kapcsolódik. Az üzenetek küldése pedig egyszerűen a *WPF* keretrendszer által biztosított kattintás kezelő metódusra van beregisztrálva, ahol a felhasználó által begépelt üzenet szintén *activity* objektumként kerül elküldésre a *Direct Channel API* használatával [41].

Ideális helyzetben ezen kívül nem szabadna egyéb logikát tartalmaznia az asztali alkalmazásnak, azonban a korábban említett okok (lásd 4.2.4. fejezet) miatt a kliensalkalmazásnak is rendelkeznie kell adatbázis-kapcsolattal ahhoz, hogy a feladatban vállalt működés megvalósulhasson. A hírek letöltésekor a chatbot csak azt a kategóriát adja vissza, amit a felhasználó kiválasztott, mert ez lehet, hogy egy *LUIS*-tól érkező előrejelzés. Majd ennek birtokában az adott kategóriára elvégez egy adatbázis lekérést a hírek táblájában és azt megjeleníti a felhasználó részére.

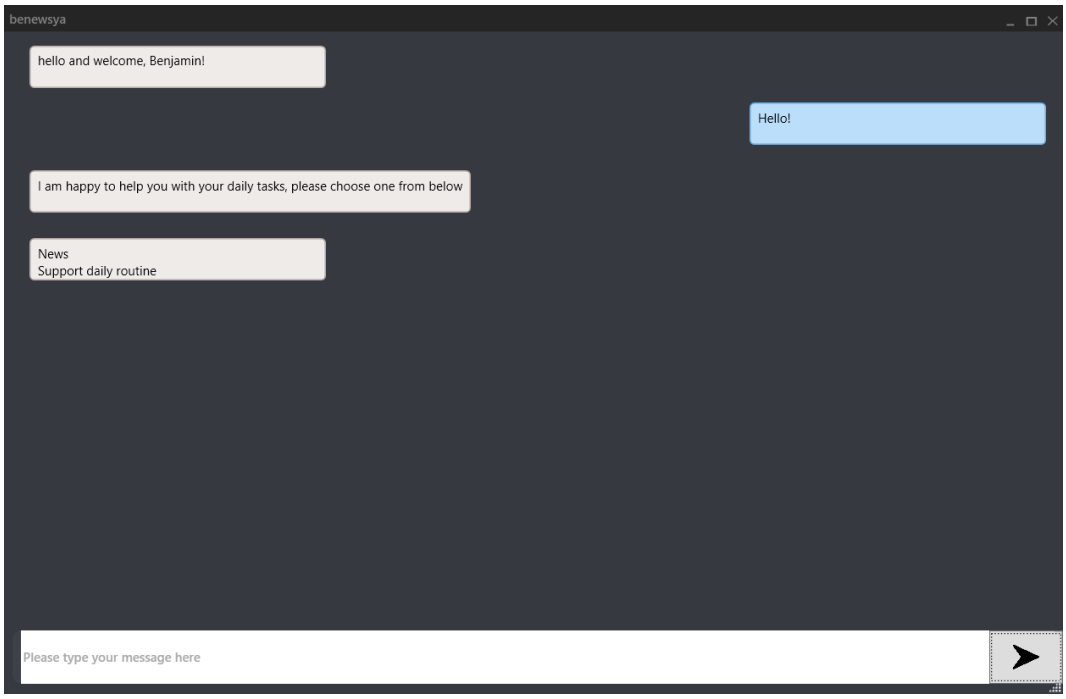
4.4. A funkcionalitások ismertetése

A dolgozat keretein belül megvalósításra került a hírfeldolgozó, valamint a napi rutin támogató modul. Ezek a rendszer fő funkcionalitásait adják, amelyek két jól elkülönített egységként kerültek implementálásra a moduláris felépítésből következően. A párbeszéd tervezésénél az egymásba egyázott vízesés párbeszéd modell került felhasználásra. Ezt a lépést megelőzi a felhasználó üdvözlése. Az alkalmazás indulásakor a rendszer vizsgálatot tesz arra vonatkozóan, hogy a felhasználó neve be lett-e már kérve korábban. Amennyiben még nem, az üdvözlés után első dolga, hogy ezt a hiányt pótolja (*4.4.1. ábra*). Majd a név megadása után a robot visszajelez, hogy a művelet sikeresen megtörtént, ezután a fő párbeszéd kezdődik el, amelyben a két fő modul közül választhat a felhasználó (*4.4.2. ábra*). Az egyes lépések közötti választás a választott lépés nevének elküldésével jelezhető a robot számára.

Ellenkező esetben a rendszer a felhasználót már a korábban megadott nevével köszönti, majd a beszélgetés elkezdéséhez egy reakció szükséges a felhasználó részéről, amelynek következtében ugyanaz a vízesés párbeszédmodell kezdődik el, mint az előzőekben (*4.4.3. ábra*).

4.4.1. ábra Üdvözlés név információ nélkül

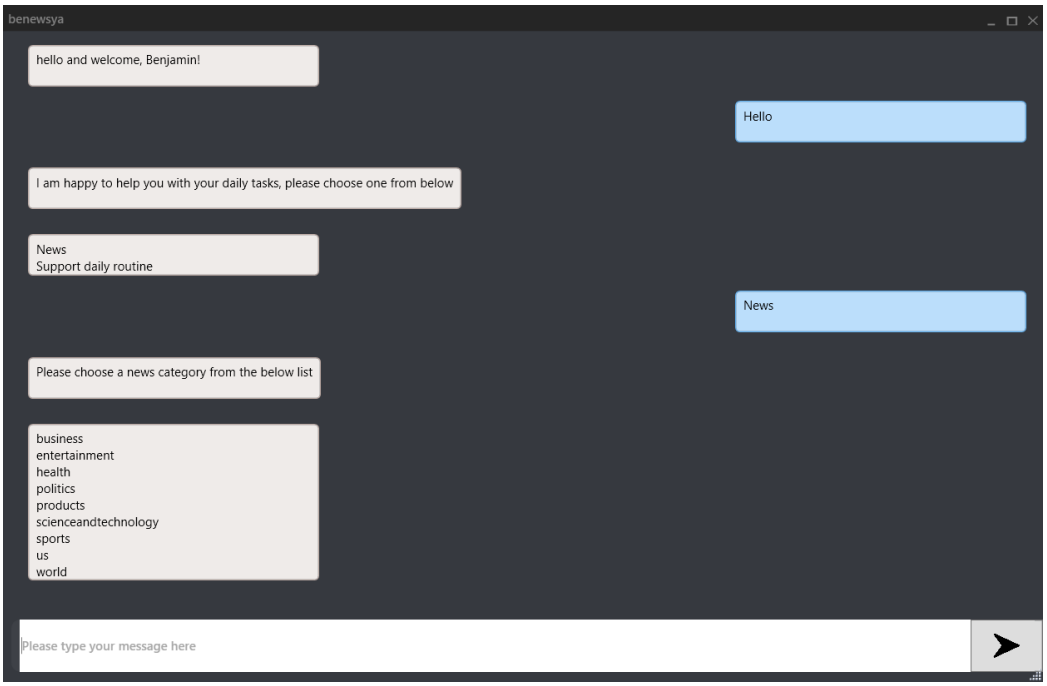
4.4.2. ábra Név nélküli felhasználó nevének bekérése



4.4.3. ábra Felhasználó köszöntése a nevével

4.4.1. Hírfeldolgozó modul

A hírfeldolgozó modult választva a lehetséges kategóriákat sorolja fel az alkalmazás (*4.4.4. ábra*), amelyből a felhasználó választhat a kategória nevének megadásával (*4.4.5. ábra*), vagy akár szabad szöveggel is (*4.4.6. ábra*).



4.4.4. ábra Hírmodul választása

4.4.5. ábra Hírkategória választása a kategória nevével

4.4.6. ábra Hír kategória választása szabad szöveggel

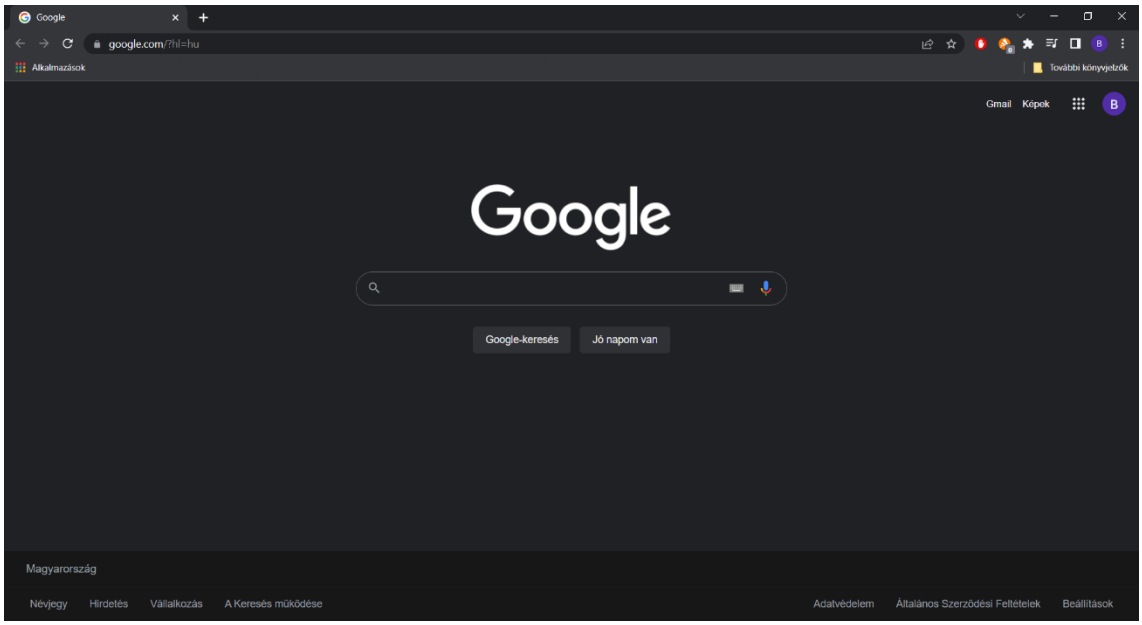
4.4.2. Napi rutin támogató modul

A napi rutin támogató modulon belül az alábbi öt kisebb alapvető feladat elvégzése került implementálásra:

- Weboldal megnyitása,
- Teendő felvétele,
- Teendők lekérdezése,
- Fejlesztői munkakörnyezet beállítása,
- Weboldal elérhetőségének ellenőrzése.

Weboldal megnyitásakor az alkalmazás bekér e felhasználótól egy weboldalt (*4.4.7. ábra*), amit először leellenőriz, hogy valóban létezik-e, amennyiben a megadott weboldal nem található, vagy érvénytelen, a kérdést addig ismétli, amíg a megadott érték érvényes nem lesz (*4.4.9. ábra*). Ellenkező esetben a megadott weboldal megnyitásra kerül az alapértelmezett böngésző segítségével.

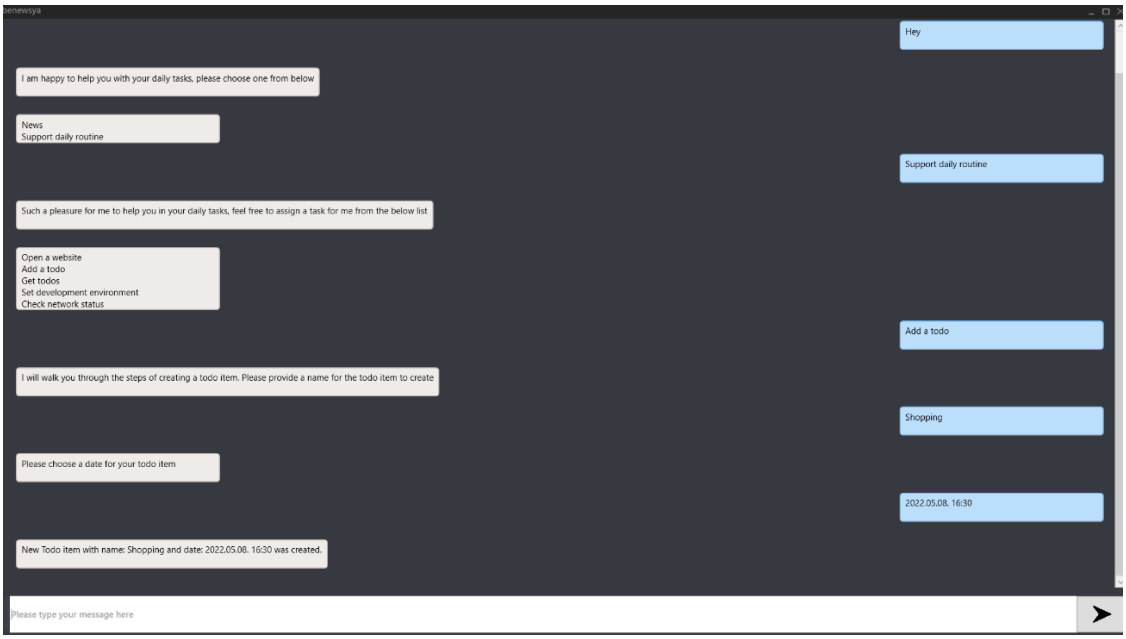
4.4.7. ábra Tetszőleges weboldal megnyitása



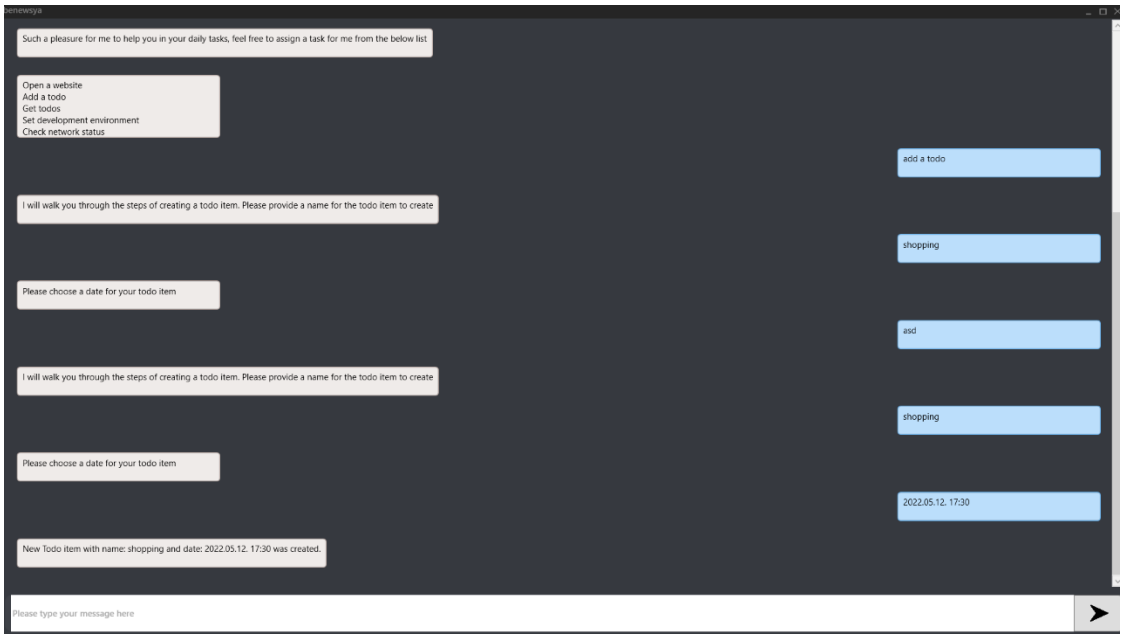
4.4.8. ábra Weboldal megnyitása eredmény

4.4.9. ábra Weboldal megnyitása hiba

Teendő felvételekor az alkalmazás a két alapvető adatot a teendő nevét és dátumát kéri be két kérdést feltéve. Amennyiben a dátum megadásakor a formátum nem megfelelő az alkalmazás nem fogadja el és újrakezdi az egész folyamatot egészen addig, amíg a formátum helyes lesz (*4.4.11. ábra*). Ellenkező esetben tájékoztat a létrehozott teendőről és kiírja az adatait (*4.4.10. ábra*).



4.4.10. ábra Teendő felvétele



4.4.11. ábra Teendő létrehozása hibás dátummal

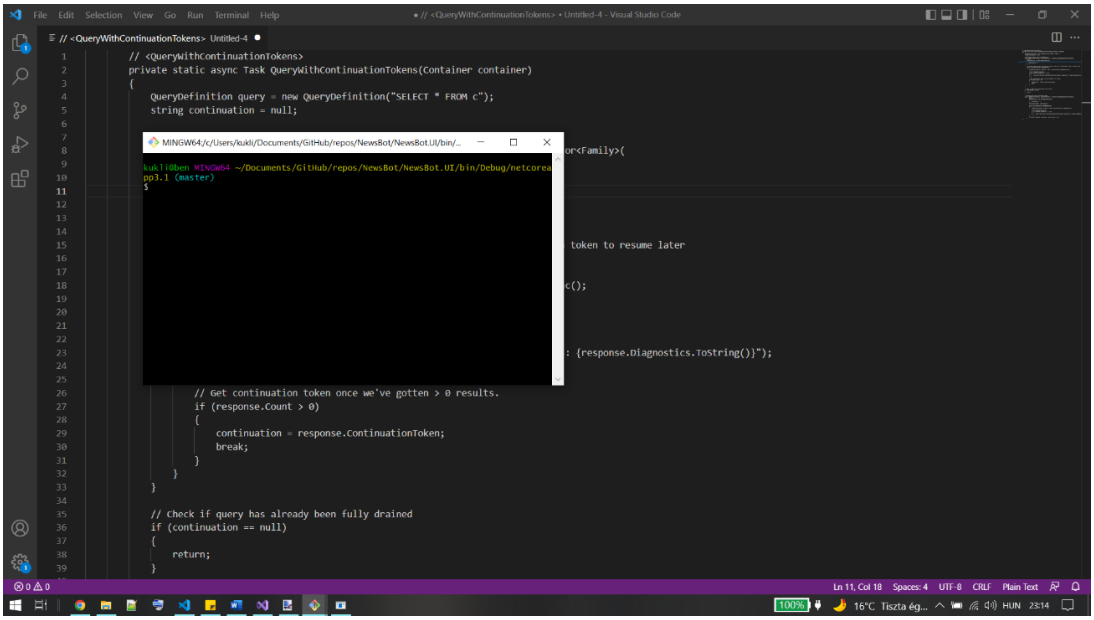
Teendők lekérésekor az alkalmazás szintén bekér egy dátumot, amelyre megvizsgálja, hogy van-e teendő felvéve, amennyiben nincs, az alkalmazás tájékoztat, hogy nem létezik teendő az adott dátumra (*4.4.12. ábra*), ellenkező esetben elküldi az adatait (*4.4.11. ábra*).

4.4.12. ábra Határidő szerint nem létező teendők lekérdezése

4.4.13. ábra Határidő szerint létező teendők lekérdezése

Munkakörnyezet beállításakor nincs külső paraméter bekérése a felhasználótól, ekkor a rendszer az *appsettings.json* fájlban megadott elérési útvonalakat nyitja meg (*4.4.13. ábra*). Ezáltal az alkalmazás által megnyitott szoftverek köre a kód újrafordítása nélkül bővíthető.

4.4.14. ábra Fejlesztői környezet beállítása



4.4.15. ábra Fejlesztői környezet beállítása eredmény

A weboldal elérhetőségének ellenőrzésekor az alkalmazás egy webcímet kér be a felhasználótól és azt vizsgálja meg, hogy az elérhető-e, mindkét esetben a megfelelő módon tájékoztatja a felhasználót (*4.4.16. ábra, 4.4.17. ábra*).

4.4.16. ábra Weboldal elérhetőségének ellenőrzése - siker

4.4.17. ábra Weboldal elérhetőségének ellenőrzése - hiba

4.5. Továbbfejlesztési lehetőségek

A rendszer számos területen továbbfejleszthető, ezek egyrészt az alkalmazás használhatóságában tesznek hozzá, másrészt olyan többletfunkcionalitásként jelennek meg, amely a dolgozat keretein belül nem volt feladat, azonban egy szélesebb rétegek által napi használatra készült alkalmazás esetén elképzelhetetlen a használatuk.

A továbbfejlesztési lehetőségek az alábbiakban merülnek ki:

- Párbeszédmodell átalakítása, a vízesés modell használatának visszaszorítása és a *deep-learning* alkalmazása minden szándék felismerésére, ezáltal javul a felhasználói élmény, mert ezáltal nincs rákényszerítve, hogy minden funkció esetén újrakezdje a beszélgetést és végig iteráljon a menüszerű beszélgetésen.
- Hírek feldolgozásánál időzített szolgáltatásként naponta többször is futhatna az RSS, amely által a felhasználó által elérhető hírek köre folyamatosan bővülne. Akár a preferenciákat is figyelembe lehetne venni, hogy esetleg azon kategóriáknál nagyobb intenzitásban valósulna meg a keresés.
- Az időzített üzenetek bevezetése lehetőséget adna arra, hogy a felhasználó értesüljön kiemelt jelentőségű hírekről azonnal, esetleg egy általa választott témában.
- Kulcsszó szerinti keresés a hírekben.
- Gyorslinkek bevezetése a felhasználó által megnyitott weboldalak esetében, a jelenlegi működés szerint a teljes elérési útvonalat kell megadnia, ami ezen segíthet, az a „*www.facebook.com*” helyett csak a *facebook* kulcsszó megoldása, amit feloldva a megadott oldalra navigál.
- Felhasználókezelés bevezetése, a BotFramework támogatja a felhasználó teljes mértékű kezelését, ehhez egy már meglévő külső identitásszolgáltató adataival lehetővé tehető a bejelentkezés, amelynek segítségével a felhasználó neve és e-mail címe is elérhetővé válik. Ez akár tovább vihető abba az irányba is, hogy a felhasználónak e-mail alapú értesítést is tudjon küldeni a robot, illetve egyéb külső szolgáltatások is hozzáadhatók, amelyből adatok lekérdezése is rendelkezésre áll. Például e-mailek, teendők, határidők lekérdezése a különböző Microsoft által készített alkalmazásokból.

5. TESZTELÉS

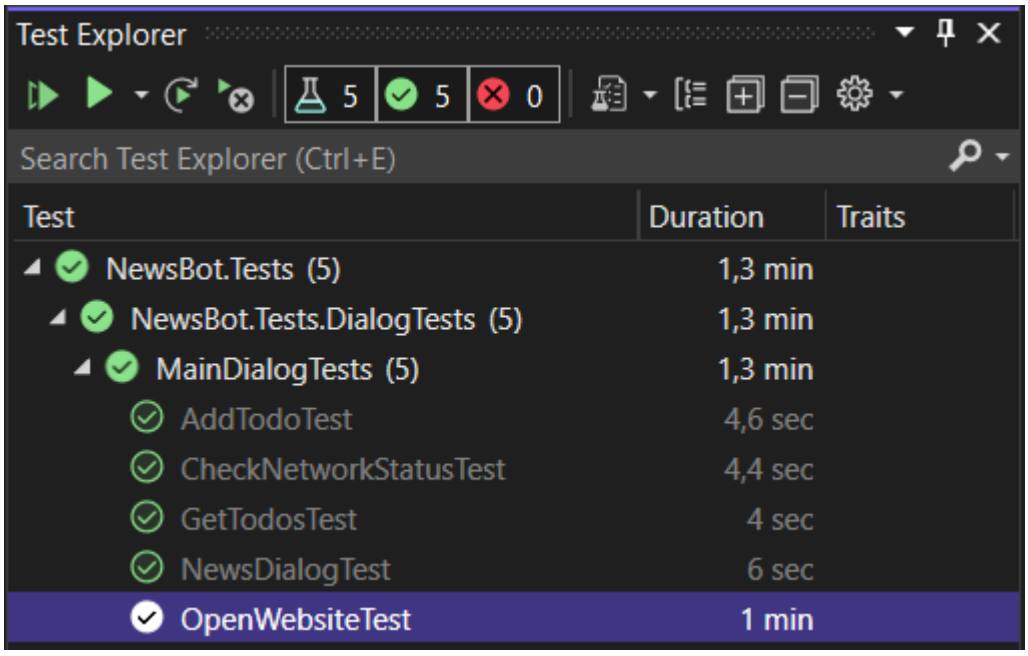
Az alkalmazás tesztelése az alábbi három módszer szerint valósult meg:

- manuális tesztelés,
- automatizált tesztelés,
- terheléses tesztelés.

A manuális tesztelés eredményei a 4.4. fejezetben kifejtettek szerint dokumentálásra került.

5.1. Automatizált tesztelés

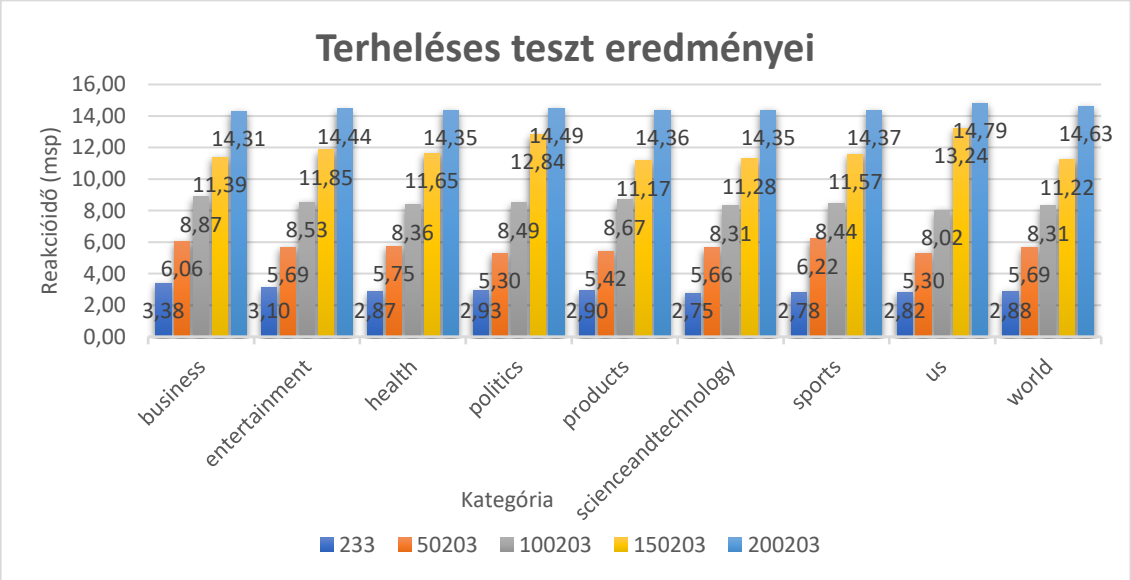
Integrációs tesztek írására is nyújt segítséget az *SDK*, a *Microsoft.Bot.Builder.Testing nuget* csomag telepítése után a lokális robot példány logikája az összes külső függőségével együtt tesztelhető. Ezzel a könyvtárral, úgy tesztelhető a csevegőalkalmazás, hogy *activity* objektumot lehet elküldeni számára, amelyben tetszőleges bemenet megadható. A megadott bemenetre a rögzített logika szerint válaszol a robot, amely aztán a megszokott módon validálható. A tesztesetek egy saját projektben kerültek implementálásra, ehhez a függőségek feloldására is szükség volt, amely a beépített *.NET Core*-ba integrált *IoC* konténer szerint történt. Az automatizált teszt eredményei igazolták a manuális tesztelés során is bizonyított eredményeket, a megfelelő válaszokat szolgáltatja az alkalmazás. A tesztesetek a dialógusokra készültek el, így 5 darab teszteset került megvalósításra (*5.1.1. ábra*).



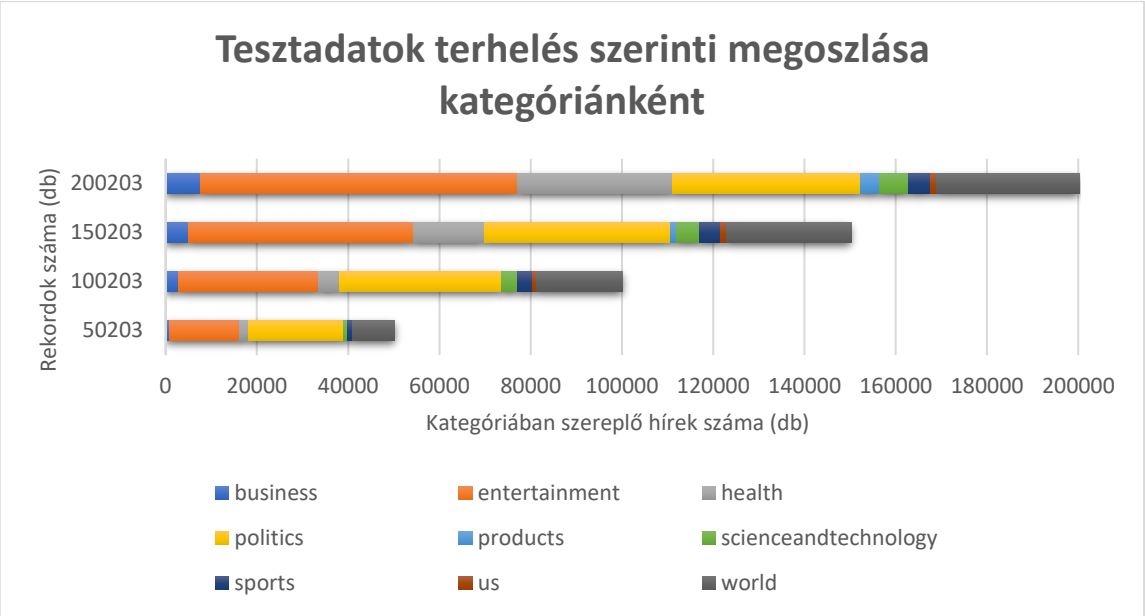
5.1.1. ábra Párbeszéd tesztesetek

5.2. Terheléses tesztelés

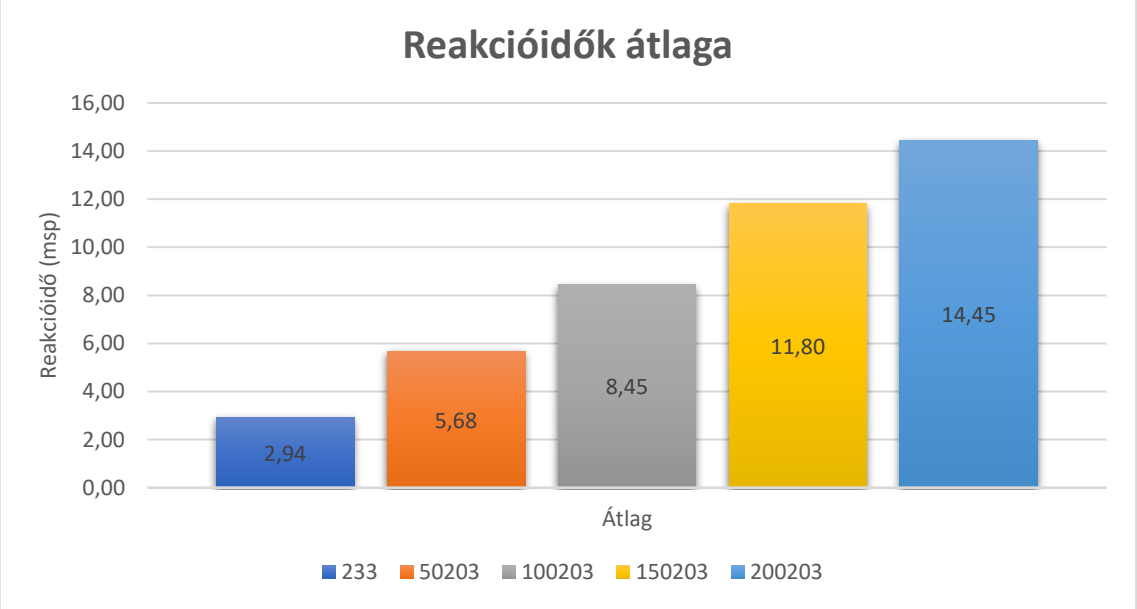
A terheléses teszt során a hírelemző modul került elemzésre, elsősorban azért, mert ez a terület tekinthető a fő funkcionalitásnak. Másrészt pedig a kliensalkalmazásban csak ezen a területen nyílik lehetőség arra, hogy a válaszidőt nyomon lehessen követni. Erre a korábban ismertetett (*4.2.4. fejezet*) okból nyílik lehetőség, az összes többi funkcionalitás esetében a válaszidők nyomon követésére nem nyílik lehetőség közvetlen módon. A felhasznált adatok internetes forrásból (https://www.kaggle.com/datasets/rmisra/news-category-dataset, *utoljára megtekintve: 2022.05.14.*) származnak, amelyben hozzávetőlegesen 200.000 hír található. A kutatómunka során fő szempont volt, hogy a forrásban található hírek rendelkezzenek kategória adattal is. Ez a *Bing API* által szolgáltatott kategóriáknak ugyan nem felel meg, viszont az adatok feldolgozásakor ezek egyértelműen beazonosíthatók voltak. A terheléses teszt során az adatok lépcsőzetes elv szerint kerültek betöltésre az adatbázisba, majd a válaszidők kerültek rögzítésre. Erre remek támogatást nyújt a *Stopwatch* osztály, amelyben az indítás és megállítás között eltelt idő egy *TimeStamp* változóba kerül eltárolásra, ezeket az adatokat felhasználva készült el az *5.2.1. ábra*. A lépésközök megközelítőleg 50.000 hírben kerültek meghatározásra, az adatok megoszlása kategóriánként teljes mértékben véletlenszerű (*5.2.1. ábra*). A tesztelés eredményei egyértelműen alátámasztják, hogy a rendelkezésre álló adatok mennyisége szabja meg a válaszidő, hiszen például a *5.2.2. ábrán* bemutatott *business* kategória elemszáma nagyságrendekkel kisebb a *politics* kategóriában szereplő hírek számánál, a válaszidők tekintetében az *5.2.1. ábra* által szemléltetett eredmények nem mutatnak számottevő eltérést. Ezt az *5.2.3. ábra* is alátámasztja, ami az átlag reakcióidőket szemlélteti kategóriánként, az *5.2.1. ábra* és *5.2.3. ábra* összevetésével bizonyítható, hogy az egyes kategóriák esetén kiugró érték nincs, azok az átlaghoz közeli értéket vesznek fel.



5.2.1. ábra Terheléses teszt eredményei



5.2.2. ábra Tesztadatok terhelés szerinti megoszlása kategóriánként



5.2.3. ábra Átlagos válaszidők kategóriánként

Tartalmi összefoglaló

A dolgozat az ember és gép közötti interakció egy új területét kutatja, középpontjában a természetes nyelv használata áll a digitális személyi asszisztensek területén, amelynek köszönhetően a felhasználó emberi nyelven szólíthatja meg a számítógépet és kérheti azt adott feladat elvégzésére.

A dolgozatban részletesen bemutatásra kerülnek a területen kiemelkedő eredményeket elért programok, amelyek a mai digitális személyi asszisztensek elődjeinek tekinthetők. Ezek az alkalmazások javarészt előre programozott válaszokat és mintakeresésen alapuló technikákat használva adtak választ a felhasználó által megadott bemenetre. A személyi asszisztensek azonban az elmúlt időszak technológiai fejlődéseinek köszönhetően sokkal szélesebb funkcionalitási kört fedhetnek le. A *deep-learning* alapú megoldások alkalmazásának kutatása és a neurális hálózat elméletének részletes bemutatása, valamint a hasonló rendszerek vizsgálata és elemzése funkcionalitás szempontjából is a dolgozat részét képezi.

Továbbá a feladat részeként a saját alkalmazás megvalósítása is bemutatásra kerül, amelyben a szakirodalmi kutatás eredményeire alapozva felhasználásra kerültek a korábbi és modern szemléletben bemutatott technológiák. A program egy modulárisan bővíthető infrastruktúrát követ, amelynek köszönhetően a későbbiekben további funkciókkal bővülhet. Az alkalmazás keretein belül hírek kerülnek feldolgozásra, egy saját hírfeldolgozó modulban, amely a híreket kategóriánként képes értelmezni. Ezen kívül egy napi rutin támogató modul is elkészült, amely a számítógép által nyújtott funkcionalitásokat használja ki, például: weboldal megnyitása, weboldal elérhetőségének tesztelése, fejlesztői környezet megnyitása.

Abstract

The thesis aims to study the area of interaction between human and machine, the focus of the paper is mainly on the natural language processing used by digital personal assistants, that enables users to practice a human language in order to communicate with the computer and make requests to carry out tasks.

The paper also thoroughly demonstrates applications that have achieved outstanding results and can be considered as predecessors of the modern personal digital assitants. These applications are mainly implemented with pre-programmed answers and pattern recognition methods to answer user inputs. However, thanks to the recent technological advancement, digital assistants are able to cover a wider range of functionalities. The thesis also demonstrates the usage of *deep-learning* solutions and shows the theory of neural networks comprehensively, furthermore contains the research of similar systems and analysis of their functionalities.

Besides the theoretical research, the implementation of a digital personal assistant is also presented, which relies on the results of the literature research and uses technologies from both the earlier, and the modern approaches. The software follows a modularly extensible infrastructure pattern, which allows further functionalities to be added at a later point. The application contains a module that is responsible for processing news and storing them with information on their categories. Besides that another module is implemented that supports the daily routine of the user by utilizing the functionalities offered by the computer e.g. opening a website, checking the availability of a website or setting up development environment.

Irodalomjegyzék

[1] Lopatovska, I.: Overview of the Intelligent Personal Assistants. *Ukrainian Journal on Library and Information Science*, 0.31866/2616-7654.3.2019.169669, *2019*, 72-79. old. (https://www.researchgate.net/publication/334351945_Overview_of_the_Intelligent_Personal_Assistants), utoljára megtekintve: 2021. 03. 14.

[2] Mahmood, Khawir & Rana, Tauseef & Raza, Abdur R.: *Singular Adaptive Multi-Role Intelligent Personal Assistant (SAM-IPA) for Human Computer Interaction*, 10.1109/ICOSST.2018.8632189, 2018, 35-41. old. (https://www.researchgate.net/publication/330878101_Singular_Adaptive_Multi-Role_Intelligent_Personal_Assistant_SAM-IPA_for_Human_Computer_Interaction), utoljára megtekintve: 2021. 03. 20.

[3] Corcoba M., Víctor & Organero, Mario.: *Artemisa: An eco-driving assistant for Android Os*. 10.1109/ICCE-Berlin.2011.6031794, *2011*. (https://www.researchgate.net/publication/252047758_Artemisa_An_eco-driving_assistant_for_Android_Os), utoljára megtekintve: 2021. 03. 27.

[4] M. Parra, J. Favela, L. Castro and A. Morales: *Monitoring Eating Behaviors for a Nutritionist E-Assistant Using Crowdsourcing*, in Computer, vol. 51, no. 03, *2018*, 43-51.old. (https://doi.ieeecomputersociety.org/10.1109/MC.2018.1731078), utoljára megtekintve: 2021. 03. 27.

[5] Du Boulay, Benedict: *Artificial Intelligence as an Effective Classroom Assistant*. IEEE Intelligent Systems. 31., 10.1109/MIS.2016.93., *2016*, 76-81.old. (http://users.sussex.ac.uk/~bend/papers/meta-reviewsIEEEV5.pdf), utoljára megtekintve: 2021. 03. 27.

[6] Stanford Encyclopedia of Philosophy: *The Turing Test.* (https://plato.stanford.edu/entries/turing-test/), utoljára megtekintve: 2021.03.28.

[7] Weizenbaum, J.: *ELIZA-a computer program for the study of natural language communication between man and machine*. Commun. ACM 9, *1966*: 36-45. old. (https://doi.org/10.1145/365153.365168https://dl.acm.org/doi/10.1145/365153.365168), utoljára megtekintve: 2021.04.02.

[8] Magnucz P. L., Baksáné Varga E., *A chatbot technológia alkalmazása magyar nyelvre, Multidiszciplináris tudományok*, 10. kötet. 2 sz., *2020*, 201-209. old. (https://ojs.uni-miskolc.hu/index.php/multi/), utoljára megtekintve: 2021.04.02.

[9] Weizenbaum J., *Computer and reason*, W.H. Freeman and Company, 1976 (http://blogs.evergreen.edu/cpat/files/2013/05/Computer-Power-and-Human-Reason.pdf), utoljára megtekintve: 2021.04.02.

[10] Pereira M. J., Coheur L., Fialho P., Ribeiro R., *Chatbots' Greetings to Human-Computer Communication*, 1609.06479, *2016*
(https://arxiv.org/abs/1609.06479), utoljára megtekintve: 2021.04.02.

[11] Mauldin, M.: *CHATTERBOTS, TINYMUDS, and the Turing Test: Entering the Loebner Prize Competition*. AAAI, *1994*.
(https://www.aaai.org/Library/AAAI/1994/aaai94-003.php), utoljára megtekintve: 2021.04.02

[12] Evgeniya Panova, *Which AI has come closest to passing the Turing test?*,
(https://dataconomy.com/2021/03/which-ai-closest-passing-turing-test/), utoljára megtekintve: 2021.04.03

[13] Shawar, Bayan & Atwell, Eric. (2007). *Chatbots: Are they Really Useful?* LDV Forum. 22., *2007*, 29-49. old.
(https://www.researchgate.net/publication/220046725_Chatbots_Are_they_Really_Useful), utoljára megtekintve: 2021.04.03.

[14] Dr. Richard S. Wallace, *The elements of AIML Style*, ALICE A. I. Foundation, Inc., *2003*
(https://files.ifi.uzh.ch/cl/hess/classes/seminare/chatbots/style.pdf), utoljára megtekintve: 2021.04.03.

[15] Csáky R.: *Deep Learning Based Chatbot Models*, 1908.08835, *2019*
(https://arxiv.org/abs/1908.08835), utoljára megtekintve: 2021.04.05.

[16] Fazekas István: *Neurális hálózatok*, Debreceni Egyetem Informatikai Kar, *2013*
(https://gyires.inf.unideb.hu/GyBITT/19/Neuralis_halozatok_v8.pdf), utoljára megtekintve: 2021.04.05.

[17] Tóth Bálint Pál: *Beszélő számítógépek mély gondolatokkal*, 2016.09.15,
(https://www.eletestudomany.hu/neuralis_halozatok), utoljára megtekintve: 2021. 04. 17.

[18] Mike Thomas: The history of deep learning: *Top moments that shaped the technology*,
(https://builtin.com/artificial-intelligence/deep-learning-history), utoljára megtekintve: 2021. 04. 17.

[19] Jianfeng G., Galley M., Li L.: *Neural Approaches to Conversational AI*, 1809.08267, *2019*
(https://arxiv.org/pdf/1809.08267.pdf), utoljára megtekintve: 2021.04.18.

[20] Christopher Olah: *Understanding LSTM Networks*
(http://colah.github.io/posts/2015-08-Understanding-LSTMs/), utoljára megtekintve: 2021.04.24.

[21] Czeba B.: *Tevékenység modellezés szenzoradatok alapján mély neurális hálózatokkal Androidos okostelefonon*, Budapesti Műszaki és Gazdaságtudományi

Egyetem, *2017* (https://tdk.bme.hu/VIK/DownloadPaper/Tevekenyseg-modellezes-szenzoradatok-alapjan1), utoljára megtekintve: 2021.04.24.

[22] Shewan D.: *10 of the Most Innovative Chatbots on the Web*, (https://www.wordstream.com/blog/ws/2017/10/04/chatbots), utoljára megtekintve: 2021.04.26.

[23] Johnson K.: *Fox News launches a Facebook Messenger bot you can't talk to*, (https://venturebeat.com/2016/08/19/fox-news-launches-a-facebook-messenger-bot-you-cant-talk-to/), utoljára megtekintve: 2021.04.26.

[24] AJ Labs: *Introducing The Al Jazeera English Facebook Bot*, (https://medium.com/@ajlabs/introducing-al-jazeera-english-facebook-bot-e1884318f394), utoljára megtekintve: 2021.04.26.

[25] Maruti Techlabs: *News Bots are Changing The Way we Read News*, (https://chatbotsmagazine.com/news-made-personal-with-chatbots-6dbba0691475), utoljára megtekintve: 2021.04.26.

[26] Mădălina Ciobanu: *With a new app and Facebook Messenger bot, Al Jazeera Media Network is expanding into digital audio*, (https://www.journalism.co.uk/news/with-a-new-app-and-facebook-messenger-bot-al-jazeera-media-network-is-expanding-into-digital-audio/s2/a701116/), utoljára megtekintve: 2021.04.26.

[27] Moses Gitari: *How to Get Siri to Read News Articles on iPhone or iPad*, (https://geeksmodo.com/how-to-get-siri-to-read-news-articles/), utoljára megtekintve: 2021.04.26.

[28] Maruti Techlabs: *News made personal with Chatbots*, (https://marutitech.com/news-made-personal-with-chatbots/), utoljára megtekintve: 2021.04.26.

[29] Martin D.: *Cortana vs. Siri vs. Google Assistant vs. Alexa*, (https://www.digitaltrends.com/computing/cortana-vs-siri-vs-google-now/), utoljára megtekintve: 2021.04.26.

[30] Kozuch K.: *Alexa vs. Google Assistant vs. Siri: Which smart assistant is best?*, (https://www.tomsguide.com/us/alexa-vs-siri-vs-google,review-4772.html), utoljára megtekintve: 2021.04.26.

[31] Nieva R.: *Google Assistant can read entire articles to you out loud*, (https://www.cnet.com/news/google-assistant-can-read-entire-articles-to-you-out-loud/), utoljára megtekintve: 2021.04.26.

[32] Sharma S.: *Activation Functions in Neural Networks*,

(https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6),

utoljára megtekintve: 2021.04.26.

[33] Hajjar A. J.: *6 Chatbot Applications / Use Cases in Healthcare in 2021*

(https://research.aimultiple.com/chatbot-healthcare/), utoljára megtekintve: 2021.04.26.

[34] Chatbots in the gaming industry

(https://www.chatbotpack.com/gaming-chatbot/), utoljára megtekintve: 2021.04.26.

[35] Az Azure-előfizetések és -szolgáltatások korlátozásai, kvótái és megkötései

(https://docs.microsoft.com/hu-hu/azure/azure-resource-manager/management/azure-subscription-service-limits), utoljára megtekintve: 2022. 05. 03.

[36] Mi a Bot Framework SDK?

(https://docs.microsoft.com/hu-hu/azure/bot-service/bot-service-overview?view=azure-bot-service-4.0) utoljára megtekintve: 2022. 05. 04.

[37] Basics of the Bot Framework Service

(https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-basics?view=azure-bot-service-4.0), utoljára megtekintve: 2022.05.05.

[38] About component and waterfall dialogs

(https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-concept-waterfall-dialogs?view=azure-bot-service-4.0) utoljára megtekintve: 2022.05.07.

[39] Managing state

(https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-concept-state?view=azure-bot-service-4.0) utoljára megtekintve: 2022.05.07.

[40] Assemblies in .NET

(https://docs.microsoft.com/en-us/dotnet/standard/assembly/) utoljára megtekintve: 2022.05.08

[41] Receive activities from the bot in Direct Line API 3.0

(https://docs.microsoft.com/en-us/azure/bot-service/rest-api/bot-framework-rest-direct-line-3-0-receive-activities?view=azure-bot-service-4.0) utoljára megtekintve: 2022.05.08.

Mellékletek

[1] Forráskód

(https://obudaiegyetem-my.sharepoint.com/:u:/g/personal/benj_stud_uni-obuda_hu/EVXbzX0w0J9Fn3vIyUu0XwgBWAvqkQ6-OpqThMOPbyoKmg?e=N10P7m)

[2] Bemutató weboldal

(https://obudaiegyetem-my.sharepoint.com/:u:/g/personal/benj_stud_uni-obuda_hu/EVXbzX0w0J9Fn3vIyUu0XwgBWAvqkQ6-OpqThMOPbyoKmg?e=N10P7m)

[3] Prezentáció

(https://obudaiegyetem-my.sharepoint.com/:u:/g/personal/benj_stud_uni-obuda_hu/EVXbzX0w0J9Fn3vIyUu0XwgBWAvqkQ6-OpqThMOPbyoKmg?e=N10P7m)