PÉCSI TUDOMÁNYEGYETEM
TERMÉSZETTUDOMÁNYI KAR
MATEMATIKAI ÉS INFORMATIKAI INTÉZET

TTKBOT CHATBOT ASSZISZTENS

Témavezető: RÉBAY VIKTOR
tanársegéd

Készítette: BARÁTH PÉTER GERGŐ
Programtervező Informatikus BSc

PÉCS, 2019

Tartalomjegyzék

I. Absztrakt

A csetbotok manapság egyre elterjedtebbek, melyeket az élet számos területén tudunk használni. Akár munkánk során is nyújthat óriási segítséget, de szórakozásból, marketing célokból és a felhasználói élmény javításából is készíthetünk ilyet. Én az utóbbi felhasználási területet választottam.

A szakdolgozat célja egy olyan csetbot készítése és hasznosságának feltárása, amely speciálisan a Természettudományi Kar honlapján, a felhasználók eligazodását segíti elő. Egy beírt szövegre válaszolva vagy egy internetes keresőhöz hasonlóan linkeket ajánl, vagy válaszol a kérdésre.

A fő célom nem az volt, hogy tökéletesen működő, minden funkciót és lehetőséget kihasználó csetbotot hozzak létre. A cél inkább megmutatni, hogy mennyi potenciál rejlik egy ilyen szoftverben.

A program megírásához használt fő nyelv a Java, melyet a rugalmassága, speciális lehetőségei és megszámlálhatatlan dokumentációja miatt választottam, ezen kívül a Java ServerFaces keretrendszer miatt, amelyet a felhasználói felület fejlesztésére használtam.

A csetbot „agya" Rivescript nyelven íródott, amely egy speciálisan csetbotokhoz kifejlesztett nyelv.

Az adatbázist PostgreSQL alapokon hoztam létre, és e nyelv segítségével is kezelem. A felhasználó felületet XHTML nyelven írtam meg, mely a ServerFaces keretrendszer egy ajánlott eleme, de ezen kívül több lehetőséget is biztosít, mint például a hagyományos HTML.

Kis mértékben megjelenik a Javascript is a programban a weboldalon lévő felhasználói élmény javítása érdekében, a CSS, amellyel a weboldalak stílusát, kinézetét szerkesztettem és a Perl, amelynek segítségével a Java nyelven keresztül is tudtam vezérelni a Rivescript nyelvet.

Kidolgozásom a főbb funkciókra helyezi a hangsúlyt, de több lehetőséget is felsorolok majd, amellyel még tovább lehetne növelni a program hasznosságát.

II. A téma kiválasztása

1. A motiváció

A program létrehozásának ötlete személyes tapasztalatból indult ki. Mikor a kar honlapján keresek információt, sokszor nagyon nehezen, vagy egyáltalán nem találom meg a keresett adatokat, ha csak a honlapon található menürendszerre támaszkodok. Ennek a folyamatnak az egyszerűsítésére általában a Google internetes keresőszolgáltatását szoktam igénybe venni. A „PTE TTK" kulcsszavak beírásával, esetleg a keresőszolgáltatás speciális keresési oldalának segítségével próbálom szűkíteni a keresési találatokat. Így általában megtalálom, amit keresek.

A további kellemetlenség akkor szokott érni, amikor több különálló információ keresése miatt ezt a metódust többször végre kell hajtsam. Ilyenkor úgy érzem, felesleges lépéseket végzek azzal, hogy egy internetes keresőt veszek igénybe, és plusz idő megy el a speciális keresésekre.

Bár több személyt ismerek, akik nem is ismerik a speciális keresési funkciókat, és sokszor panaszkodnak, hogy az internetes keresőkkel is nehezen találják a keresett információt. Így lehetséges, hogy felhasználók többsége is hasonló problémával küzd.

1.1. Speciális kereséssel kapcsolatos felmérés kérdései, eredmények és konklúzió

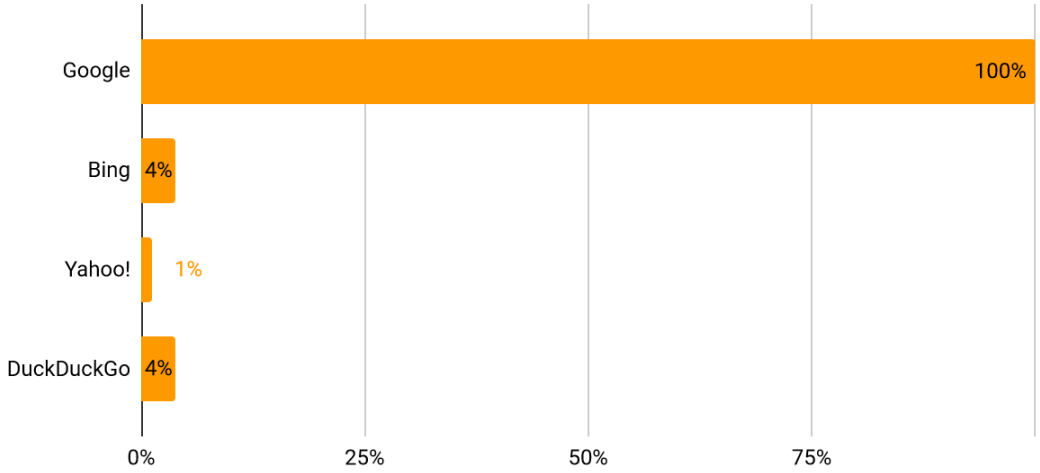
A feltevésemet egy felméréssel próbáltam igazolni. A kérdések a speciális keresési lehetőségek használatára és ismertségének arányára irányultak.

A felmérést a Google Form[1] szolgáltatásával készítettem *Speciális keresés használata az interneten*[2] címmel. Összesen 77 válasz érkezett. Az alábbiakban az egyes kérdésekre kapott eredményekre térek ki.

[1] https://www.google.hu/intl/hu/forms/about/
[2] https://forms.gle/9EwsRE5H3ZMP8DJF7

1.2. Ön milyen internetes keresőt/keresőket használ?

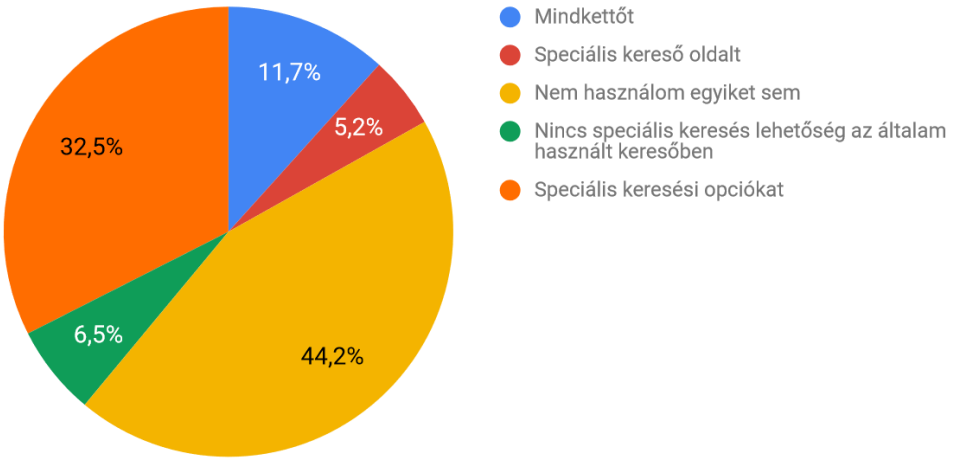
Őszintén szólva az 1. ábrán látható eredmény nem lepett meg. Kivétel nélkül, mindenki a felmérés kitöltői közül használ Google keresőszolgáltatást, ami azt jelenti, hogy biztosan számos lehetőségük van a speciális keresésre. Bár a többi keresőszolgáltatásnak is vannak ilyen lehetőségei, de nem olyan sok, mint a Google-nak.



1. ábra: Keresőszolgáltatások használata

1.3. Ön használja az adott kereső speciális keresési opcióit vagy speciális keresési feltételek megadására létrehozott külön oldalát?

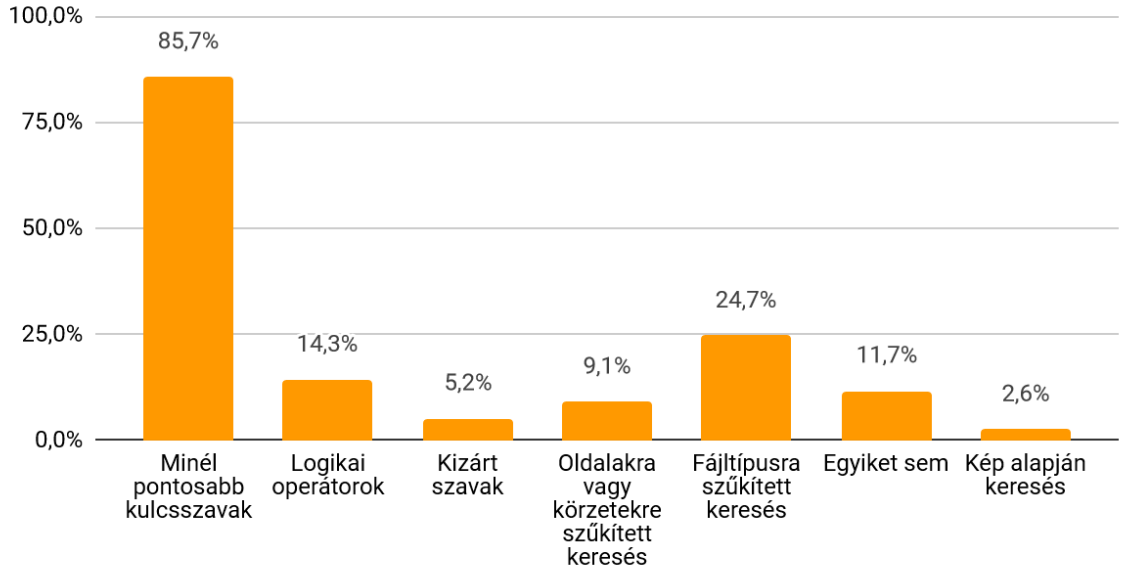
Az eredmény érdekes, hiszen 6,5%-a kitöltők közül azt mondta nincs ilyen lehetőség. Pedig a Google tartalmazza mind a kettő speciális keresési módot. Ez azt jelenti, hogy ezek a felhasználók nem is tudnak erről a lehetőségről. Kicsit kevesebb, mint a megkérdezettek fele pedig azt mondta, hogy nem használja ezeket.



2. ábra: Keresési opciók használatának aránya

1.4. Ha ön használja a Google internetes keresőt, milyen módon szűkíti a keresési találatokat?

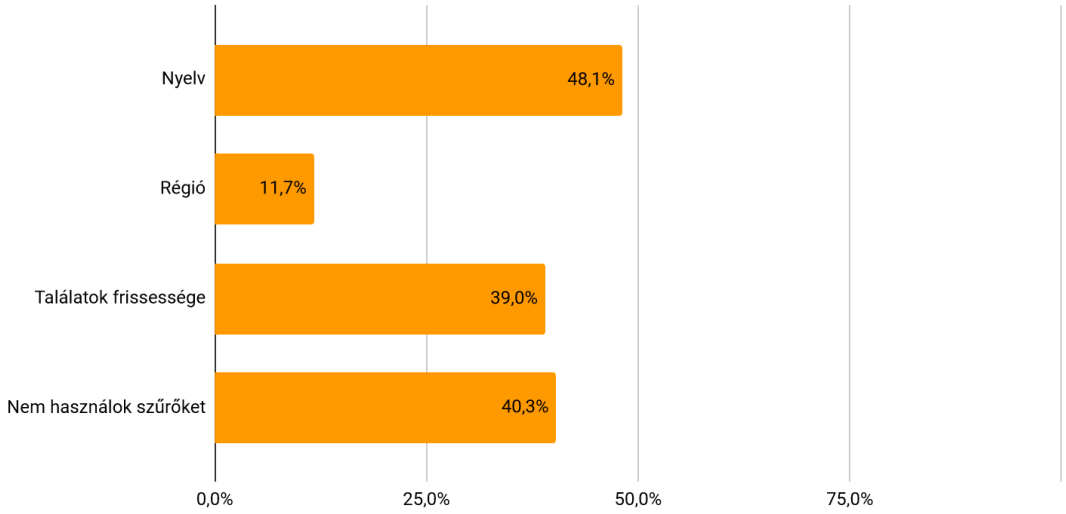
A diagramm alapján jól kiolvasható, hogy nagy százalékuk a kitöltőknek csupán minél pontosabb kulcsszavakat alkalmaz. Ezzel a módszerrel én általában jó eredménnyel megtalálom, amit keresek, de nem mindig.



3. ábra: Keresés szűkítés módjainak használati aránya

1.5. Ha ön használja a Google internetes keresőt, milyen szűrőket használ általában?

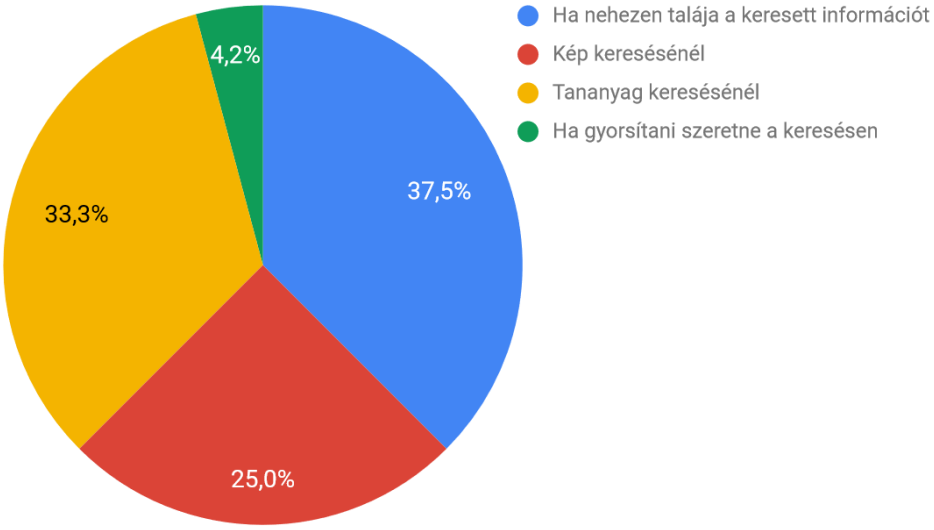
A legtöbbet használt szűrő a nyelvi szűrő, ami arra enged következtetni, hogy többször a nem megfelelő nyelvű weboldalak is a találati listára kerülnek az adott keresésénél.



4. ábra: Szűrési módszerek használati aránya

1.6. Ha ön használ speciális keresést, milyen esetekben használja azt?

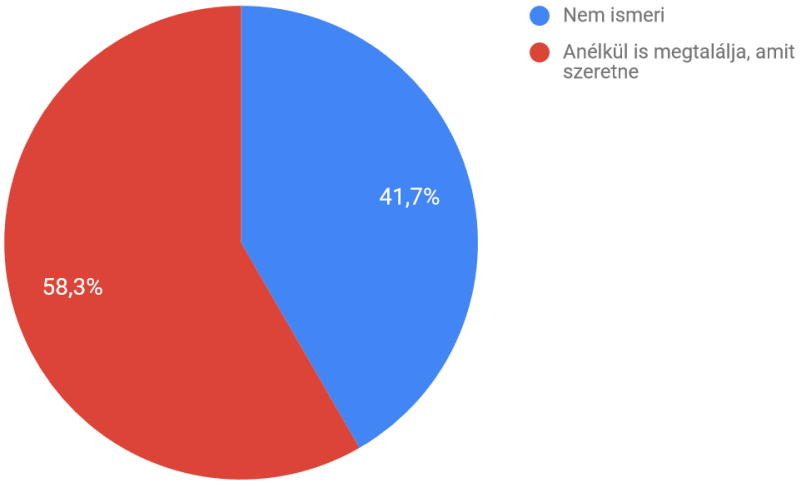
A kérdőív erre a kérdésre egyénileg beírt választ várt, így nem írom le az összeset, de különböző típusokra szedve bemutatom azokat.



5. ábra: Speciális keresési esetek aránya

1.7. Ha ön nem gyakran vagy egyáltalán nem használ speciális keresést, mi ennek az oka?

Ugyancsak egyéni válaszadási lehetőség volt, így itt is az összesített eredményt mutatom be. A felmértek közül több mint 40%-a nem ismeri a speciális lehetőségeket.



6. ábra: A speciális keresés mellőzés okainak aránya

1.8. Konklúzió

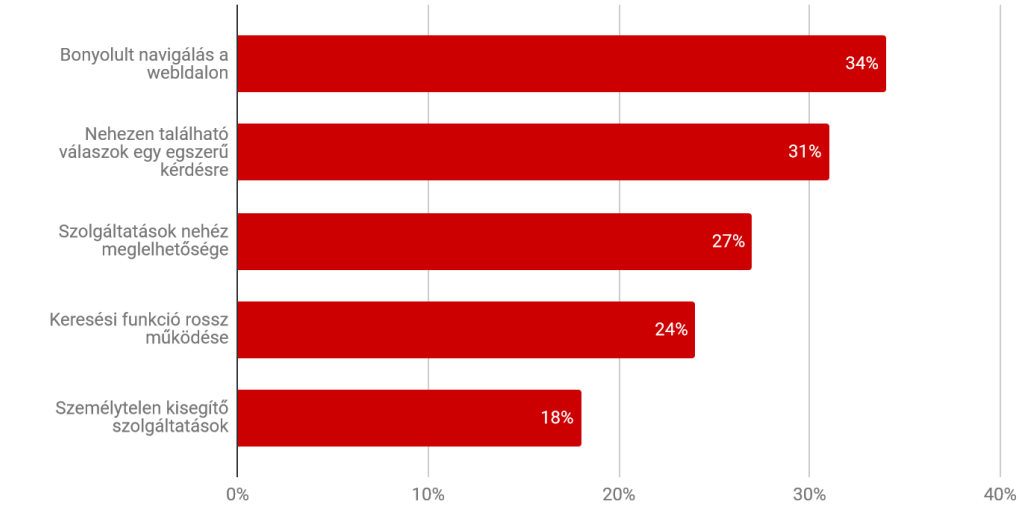
Arra az álláspontra jutottam, hogy egy csetbot, mely a kar honlapján van implementálva, biztosan speciálisan a honlapon keresne, így kikerüli az internetes keresők használatát és az ezzel kapcsolatos nehézségeket. Ezen túlmenően pedig sok más lehetőséget nyújtana, amelyet az internetes kereső szolgáltatások nem nyújtanának. A további lehetőségekkel az utolsó fejezetben foglalkozok.

2. Csetbotokkal kapcsolatos felmérés kérdései, eredmények és konklúzió

Arra is kíváncsi lettem, mekkora igény lenne egy ilyen csetbotra. Hallottam pozitív és negatív véleményeket is ezzel kapcsolatban, így hát utána néztem a dolognak. Arra voltam kíváncsi melyek azok a tulajdonságok, amik miatt ajánlják vagy óvnak a csetbotoktól.
Egy 2017 november 6.-i, az Amerikai Egyesült Államokban végzett felmérést alapul véve végzem e kérdés elemzését. A felmérést 1051, 18 és 64 év közötti személyeken végeztek. (Drift, SurveyMonkey, Audience, myclever & Salesforce, 2018)
Bár a felmérés főleg üzleti oldalról közelíttette meg a kérdést, néhány felmerülő probléma általánosságban is hasznosnak bizonyult. Így most csak ezekre a nézőpontokra szorítkozok.

2.1. Milyen frusztrációk érték önt az utóbbi egy hónapban a weboldalakon?

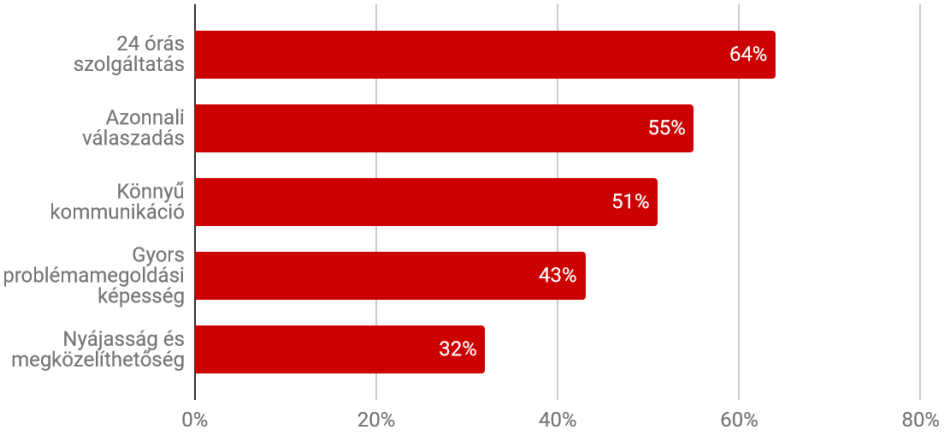
Legtöbben ugyanazzal a problémával találkoztak, amivel én a kar honlapjával kapcsolatban. Egy, a honlapra megírt csetbot, úgy gondolom megoldást jelentene a felmérésben felmerülő összes „frusztrációra”.



7. ábra: Tapasztalt problémák az online világban
(Drift, SurveyMonkey, Audience, myclever & Salesforce, 2018)

2.2. Ha elérhető volt (és jól működött) egy webszolgáltatás chatbotja, mik voltak azok az előnyök, amiket elvárt a bottól?

Ezen tulajdonságokat mind birtokolja a csetbot. Természetesen ha azt megfelelően leprogramozták.

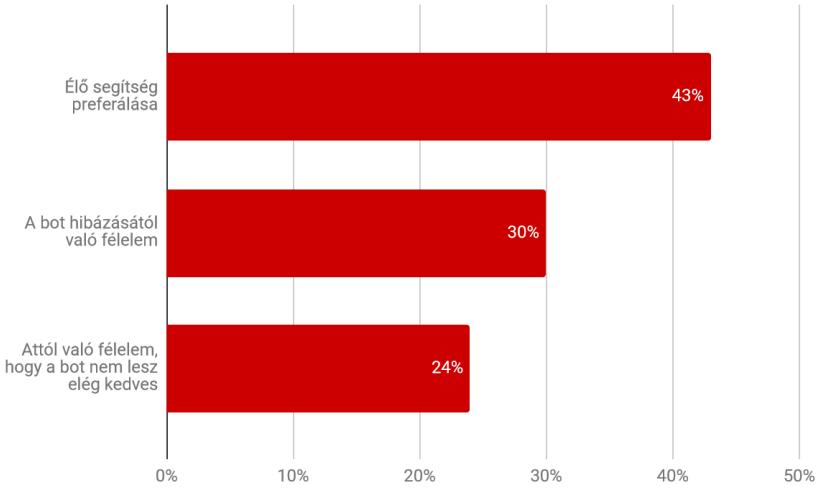


8. ábra: A csetbot lehetséges előnyei

(Drift, SurveyMonkey, Audience, myclever & Salesforce, 2018)

2.3. Mi akadályozná meg a csetbot használatában?

A 9. ábrán látható válaszok közül a harmadikkal, amely a bot kedvességét említi, mindenképp lehet kezdeni valamit. Csak olyan személyiséget kell adni a csetbotnak, melyet az emberek kedvesnek tartanak. De ezt követően a második pont már a program megírásának minőségétől függ, az első pont pedig már nem a csetboton múlik. De úgy gondolom, pont emiatt, mindig kell lehetőséget biztosítani egy weboldalnak, arra, hogy a felhasználó tudjon elő személyel beszélni.



9. ábra: Potenciális akadályok a csetbot használatával

(Drift, SurveyMonkey, Audience, myclever & Salesforce, 2018)

2.4. Konklúzió

Az eredményeket alapul véve, bátran indulhattam ki abból, hogy a csetbot jó megoldás lehet a problémára, amelyet a felhasználók is szívesen használnának.

3. Lehetőségek

A megoldás megtalálása után azt kellett eldöntsem, hogy milyen legyen a bot mechanikája. Válaszadás szempontjából három féle mechanikát különböztettem meg. Ezek az előre meghatározott válasz szerinti, kulcsszó és kontextus alapú mechanikák. (Phillips, 2018)

3.1. Előre meghatározott válasz mechanika

Más néven menü- vagy gombalapú csetbotok. Az ilyen botok bizonyos válaszlehetőségeket kínálnak fel, amelyek alapján a felhasználó dönthet melyiket választja. Ezek a beszélgetések egy megadott útvonalon haladnak. Egy adott válasz gombot megnyomva a felhasználó megkapja az adott kérdésre a választ és az esetleges további kérdéslehetőségeket. (Phillips, 2018)

Ilyen például a Messenger csetbot kezelő rendszerének egyik jó példája a Wall Street Journal botja[3] vagy az LG Televízió vásárlást segítő asszisztense[4].

3.2. Kulcsszó alapú mechanika

Ebben a mechanikában, is lehetnek meghatározott válaszok, de a bot már a felhasználó által beírt szövegre reagál, a beírt szövegből felismert kulcsszavak alapján. (Phillips, 2018)

Erre jó példa a Messengeren található Agymozgató[5] nevű bot.

3.3. Kontextus felismerés mechanika

A három közül messze ez a mechanika a legfejlettebb. Ennél a típusnál a csetbot a gépi tanulást és a mesterséges intelligenciát használja ahhoz, hogy önmagát fejlesztve egyre „okosabb” legyen és egyre pontosabb válaszokat tudjon adni a felhasználónak. (Phillips, 2018)

Egy régebbi magyar példa erre az általános célból készült tanuló bot, Lafoxka[6]. De akár a mai gigacégek csetbotjaira is gondolhatunk Ilyen például az Apple cég Siri nevű botja vagy

[3] https://www.facebook.com/messages/t/wsj
[4] https://www.lg.com/hu/televiziok/osszes-televiziok
[5] https://www.facebook.com/messages/t/agymozgato
[6] http://www.lafoxka.hu/

a Google cég Google Asszisztense. Bár az utóbbi kettő még nem érhető el magyar nyelven.

3.4. Célkitűzés

Az első mechanika, úgy gondolom nem lenne célszerű erre a problémára. Kvázi a csetbot ebben az esetben barkóbázik a felhasználóval, amely egy speciális kérdésnél lassú válaszadást eredményezhet. A gépi tanulást és a mesterséges intelligenciát pedig úgy vélem felesleges egy ilyen egyszerű felhasználású bothoz igénybe venni. A legcélszerűbbnek, a második, kulcsszó alapú mechanikát tartottam, így ezt választottam ki a csetbot elkészítésének módjának.

A célkitűzésem így egy olyan csetbot készítése lett, amely bármely üzenetből a kiolvasott kulcsszavak alapján képes reagálni arra és minimális mértékben csevegni is tud a felhasználóval.

A fő funkciója, hogy segítsen a beírt kérdés alapján, a PTE TTK honlapján és az ezen honlapjáról kiinduló linkek közül megkeresni a releváns weboldalakat.

Ezeket a találatokat link formájában kapja meg a felhasználó. Ehhez természetesen a weboldalon megjelenő információkat is le kellett mentsem, ahonnan a bot ki tudja olvasni a keresett információt. Így az adatok kiolvasása, rendszerezése és a kulcsszavak kiszűrése is célom lett.

Bár a maximális hasznosságot akkor érné el a program, ha az eredeti weboldalon tudtam volna implementálni, de sajnos erre nem volt lehetőségem. Így egy demo oldalt hoztam létre, ahol szemléltetni tudtam a csetbot működését. Az oldal a https://bpg-ttkbot.herokuapp.com/chatbot.xhtml linken érhető el.

III. Eszközök

1. Rivescript

A Rivescript egy nyílt forráskódú, script nyelv, mellyel képesek vagyunk egy csetbot intelligenciáját kezelni, megírni. Egyszerű szintaxisa és könnyű kezelhetősége miatt tökéletes választás volt számomra. A Rivescript nem teljes, all-in-one csetbot nyelv. Önálló szoftverkönyvtárként van kialakítva, amely egy egyszerű API-val bármilyen meglévő kódbázishoz csatlakoztatható. Ilyen kódbázis lehet például Python, Go, de akár a Java nyelven is megírva. A megalkotásának célja az volt, hogy könnyen tanulható legyen a fejlesztőknek, a többi csetbot nyelvvel ellentétben, amelynél például bonyolult XML kódokat kellene használni (mint például az AIML-nél), vagy sokkal bonyolultabb szintaxist kellene megtanulni (mint, például a ChatScript nyelvnél). (Petherbridge, é. n.)
Nagy pozitívum a nyelvben a fentebb említett önálló szoftverkönyvtár használatának rugalmassága. A Rivescript nagyobb szabadságot ad a többi hasonló nyelvhez képest. A többi már egy kidolgozott keretrendszerrel és struktúrával rendelkezik, ami bizonyos tekintetben jó, de ha nincs rá szükség, felesleges, esetleg lecsökkentheti a lehetőségek számát. Ebben az esetben a kiválasztott nyelven mi írjuk meg például a kezelőfelületet vagy az üzenetekre kapott válaszok megjelenítését.
A nyelv bemutatásánál csak a legfontosabb dolgokra koncentráltam. A példák a nyelv egyszerűségét hivatottak bemutatni.

1.1. A nyelv felépítése

A Rivescript egyszerű kérdés-válasz párokból épül fel, melyek egy vagy több *.rive* kiterjesztésű fájlban tárolódnak el. Az 1. forráskódban egy egyszerű példát mutatok erre:

```
+ helló bot
- Helló, Ember!
```

1. forráskód: Alapok

A ’+’ jel utáni sor az úgynevezett trigger. Ez a sor, amelyben azonosságokat keres a Rivescript fordítója azzal, amit a felhasználó ír be az input mezőbe. A trigger csak kisbetűket, számokat és az aposztróf közötti karaktereket tartalmazhatja: ’*(|) [ ] * _ # { } < > = /*’. Viszont a felhasználó által beírt szövegben automatikusan figyelmen kívül hagyja a nagybetűket és a felsorolásból kihagyott karaktereket. Ebben a példában, ha a bot a

következő üzenetet kapja: „Helló Bot!”, ő ezzel válaszol rá: „Helló, Ember!”. Még akkor is, ha a triggerben kisbetűvel van írva a szöveg és nincs felkiáltójel. Természetesen előbb egy paranccsal meg kell adjuk, hogy UTF8 betűkészlettel dolgozzon, ha ékezetes betűket is szeretnénk használni. (Petherbridge, é. n.)

1.2. A nyelv használata

A nyelv használatára két lehetőség van. Az első az online tesztelés[7]. Ebben ez esetben egy előre létrehozott felületen lehet kipróbálni a nyelvet. A második lehetőség, az úgynevezett interpreter, tolmács megírása. Ez összeköttetést biztosít a Rivescript és azon nyelv között, amellyel kezelni szeretnénk a programot. Szerencsére Java nyelven megtalálható az előre megírt tolmács és a hozzá tartozó egyéb osztályok, így ezzel nem kellett foglalkozzak. (Petherbridge, é. n.)

1.3. Felépítése:

A ’!’ szimbólum egy definíciós szimbólum. E szimbólum után bizonyos kulcsszavakat írva van lehetőség botra vonatkozó változók, helyettesítő szavak definiálására vagy Rivescript verziójának megadására stb. Ezek kerülnek mindig a fájl legelejére. Mint sok más nyelven a „//” szimbólum párral lehet kommenteket írni, melyeket nem vesz figyelembe a fordító. A definíciók után írjuk a triggereket és a hozzájuk tartozó válaszokat. A különböző kérdés-válasz szekciók dupla sortöréssel kell elválasztani egymástól. Ezek után pedig az esetleges makrókat lehet definiálni. (Petherbridge, é. n.)

```
! version = 2.0

// bot globális változói
! var name = TTKBot

// Substitutions - cserék
! sub mijen = milyen

+ milyen szép időnk van
- Ahogy mondod!
```

2. forráskód: Rivescript felépítése

[7] https://play.Rivescript.com/

Ezen csere definiálása után, ha a felhasználó a következőt írná a bemenetbe: „Mijen (sic!) szép időnk van!” A bot így is felismerné, és a válasz ez lenne: „Ahogy mondod!”.

1.4. Több lehetőség megadása

Több válasz írására is van lehetőség egy triggerhez, amelynek módja az, hogy több ’-’ szimbólummal kezdődő sort írunk egymás alá. Ennek eredménye az, hogy véletlenszerűen választ a program a válaszok közül. Természetesen a súlyozásra is van lehetőség. Ha valamely válasz nem rendelkezik meghatározott súllyal, az alapértelmezetten 1 súlyt jelent. A súlyok összeadódnak és az összsúly és a megadott súly közötti százalékos érték adja meg, mennyi eséllyel jelenik meg az adott válasz. Ezen kívül azonos válaszokhoz rendelt trigger is lehet többféle. Zárójelbe téve és ’|’ szimbólummal elválasztva adhatjuk meg ezeket a különböző lehetőségeket. A szögletes zárójelben lévő szó opcionális, vagyis az abban lévő szó beírása nélkül is érti és reagál rá a csetbot. (Petherbridge, é. n.)

```
+ [nagyon] (szeretem|imádom|kedvelem) pécset
- Én is!{weight=20}
- Ezt öröm hallani!{weight=25}
- Me too!
```

3. forráskód: Lehetséges bemenetek és válaszok súlyozása

A súlyok összege ebben az esetben (20+25+1=46), így például az első válasz lehetősége a megjelenésre ((20/46) •100 ≈43,48%).
A súlyokat kapcsos zárójelek közé kell írni a `weight` kulcsszó után. Az ezen szimbólumokban használt kulcsszavakat tag-eknek nevezzük. Ezekre később térek majd ki.

1.5. Nyílt triggerek

Van lehetőség nyílt triggerek használatára, amelynek célja, hogy adjon valamennyi szabadságot a felhasználónak vagy az, hogy reagáljon a bot a felhasználó által beírt speciális információkra. A ’*’, egy helyettesítő karakter, amely egy szót takar magában. Akár több helyettesítő karakter használatára is van lehetőség. Ha szögletes zárójelek közé tesszük, figyelmen kívül hagyja az ezen helyén megjelenő bármekkora, bármennyi szóból álló szöveget. A legutolsó példa, ha csak egy ’*’ van a triggerben. Ez „elkap” minden olyan inputot, amihez nem tartozik válasz. (Petherbridge, é. n.)
A nyílt triggerekre a 4. forráskódban mutatok példát.

```
+ a kedvenc színem a *
- A <star> nagyon szép szín

+ a kedvenc színem a * és a *
- A <star1> nagyon szép szín, de a <star2> nem a kedvencem

+ [*]fityisz[*]
- Ne mondj ilyen csúnyát!

+ *
- Ezt sajnos ezt nem értem!
```

4. forráskód: Nyílt triggerek

1.6. Változók és tömbök

Mint minden programozási nyelvben, itt is van lehetőség változók használatára, ami elengedhetetlen különböző információk eltárolásához. Az 5. forráskódban kívánom ezt szemléltetni. A `set` tag-el deklarálunk egy, az adott felhasználóra vonatkozó változót és azt a kezdőértéket adjuk neki, amelyet a felhasználó beírt. A `get változónév` segítségével pedig visszakapjuk ezt a változót. A fájl legelején definiált ugyanilyen nevű változó a botra vonatkozik és `<bot name>` tag-el tudunk hivatkozni rá. (Petherbridge, é. n.)

```
! var age = 3

+ én * éves vagyok
- <set age=<star>>Az egy szép kor.

//Reagál arra, ha még nem kapott adatot az adott változóhoz
+ menyi idős vagyok
* <get name> == undefined => Még nem mondtad meg ezt nekem.
- A korod: <get age>!

// If is létezik a nyelvben
+ én tudok jelentkezni az egyetemre
* <get age> == undefined => Sajnos nem tudom mennyi idős vagy.
* <get age> < 18 => Sajnos 18 évesnek kell legyél minimum.
- Persze, 18 év fölött lehet jelentkezni.

+ te hány éves vagy
- Én még csak <bot age> éves vagyok.
```

5. forráskód: Változók használata

Tömbök definiálására is van lehetőség. Csak szóközzel vagy ’|’ karakterrel kell elválasztani a tömb elemeit, attól függően, hogy egy szavas vagy több szavas elemeket szeretnénk-e eltárolni. (Petherbridge, é. n.)

```
! array colors = piros zöld kék sárga

// Ha több szó szerepel egy adatban
! array kék = zöldes-kék|kékes szürke|olyan kék, mint a tenger
```

6. forráskód: Tömbök definiálása

A triggerben pedig a `(@colors)`-al lehet hivatkozni a tömbre. Így, ha bármely, a tömbben szereplő színt írja be az inputba a felhasználó, a bot reagál rá. (Petherbridge, é. n.)

1.7. Irányított beszélgetések

Ezek olyan beszélgetésrészletek, amelyek több kérdés-válasz párból állnak. Ilyen lehet például, amikor a bot visszakérdez és arra reagál.

```
+ (nem szeretem|utálom) az egyetemet
- Kifejtenéd kérlek, miért mondod ezt?

+ *
% kifejtenéd kérlek miért mondod ezt
- Köszönöm, hogy ezt elmondtad! <set hate=<star>>
```

7. forráskód: Irányított beszélgetések

Fontos, hogy a ’%’ utáni szöveg kisbetűkkel és a tiltott írásjelek nélkül jelenjen meg. A választ eltárolva egy változóba, lehetőségünk van azt elmenteni például egy adatbázisba és felhasználni adatgyűjtésre.

1.8. Témakörök (Topics)

Ezzel a lehetőséggel beszélgetési ciklusok létrehozására van lehetőség, amelyből csak a megfelelő válasz megadásával léphet ki a felhasználó. Egy személyiséggel felruházott csetbotnál például ez jó megoldás lehet egy sértés kezelésére.

```
+ utállak
- Komolyan? Nem beszélek veled, amíg bocsánatot nem kérsz!{topic=sorry}

> topic sorry

 + (bocsi|sajnálom|elnézést|bocsánat|bocsánatot) [*]
 - Rendben, megbocsátva! {topic=random}

 + *
 - Nem válaszolok, amíg nem kérsz bocsánatot!

< topic
```

8. forráskód: Topics

Az alapértelmezett témakör a random, ezzel a témakör nélküli szekcióba ugrik a program. Hasonló elven létezik a „begin topic”, amivel beállítható, hogy mik legyenek azok a válaszok, amivel a csetbot elkezdi a beszélgetést, akármilyen inputtal is találkozna. (Petherbridge, é. n.)

1.9. Tagek

A tageket kapcsos zárójelek közé írt parancsok, amelyeknek kezdő és befejező tag-jük van. Több ilyet is bemutattam már a példák között, mint például: `star, get, set, weight` stb. Ezek, mint beépített függvények működnek. A 9. forráskódban például a **star** helyébe beírt szónak a kezdőbetűje nagy, míg a többi kicsi lesz.

```
+ a nevem *
- <set name={formal}<star>{/formal} >Örülök a találkozásnak, <get name>!
```

9. forráskód: Formal tag

Egyszerűsíteni is lehet ezt a parancsot, amely funkcióját tekintve ugyanazt csinálja, mint a 9. forráskódban bemutatott példa.

```
+ a nevem *
- <set name=<formal> >Örülök a találkozásnak, <get name>!
```

10. forráskód: Egyszerűsített formal tag

Ezen kívül matematikai műveleteket is lehet használni *add*, *subtract*, *multiply*, *divide* tag-ekkel, vagy például használható még a `<reply>` tag, amellyel az előző válasz kapható meg. (Petherbridge, é. n.)

Fontos megemlíteni még a `<call>` tag-et, amivel a belső vagy külső függvényeket, az úgynevezett makrókat tudjuk meghívni. A makrók, a fájlban meghívható, nem Rivescript nyelven megírt függvények vagy metódusok. Ez teszi lehetővé, hogy tovább bővíthessük a nyelv eszközkészletét. Az alábbi példában a Google nevű metódus kerül meghívásra, ahol a star, a metódus attribútuma. (Petherbridge, é. n.)
Bár ez nem egy beépített metódus, úgyhogy használat előtt meg kellene írni azt.

```
+ google: *
- <call>google<star></call>
```

11. forráskód: Makró példa

2. Maven

Ennek a technológiának a segítségével többek között, úgynevezett dependenciákat, függőségeket lehet létrehozni, amellyel nagyban megkönnyíthető a programozói munka. Ezek a függőségek `.pom` kiterjesztésű fájlban vannak elhelyezve XML nyelven megírva. Ez a technológia leveszi a programozó válláról a különböző osztályok beimportálási feladatát. Ezt a függőségek automatikusan megteszik. (The Apache Software, é. n.)
A 12. forráskód például a com.Rivescript osztály dependenciája.

```
<dependency>
	<groupId>com.Rivescript</groupId>
	<artifactId>Rivescript-core</artifactId>
	<version>0.10.0</version>
</dependency>
```

12. forráskód: Maven példa

3. PostgreSQL

A PostgreSQL egy nagy teljesítményű, nyílt forráskódú relációs adatbáziskezelő nyelv, amely az SQL nyelven alapszik. Bár elődjéhez képest a szintaxisa kisebb mértékben különbözik tőle, de funkciója ugyanaz, sőt több lehetőségünk is van e nyelv használatával.
Ezen kívül biztosítva volt számomra egy szerverkörnyezet is, a PGAdmin 4.6[8]. Ennek a segítségével a munkára használt számítógépemen tudtam tesztelni a programom szerver oldali feladatait de ezen keresztül vezérlem a felhő szolgáltatásban igénybe vett adatbázist is. (The PostgreSQL Global Development Group, é. n.)

[8] https://www.pgadmin.org/

4. Java és Java ServerFaces (JSF)

A TTKBot, egy Java ServerFaces keretrendszerrel készült webalkalmazás. Spring Boot-ot használ, amely olyan Java eszköz, amellyel a létrehozott JAR fájlokat tudjuk Tomcat szerveren futtatni. A keretrendszer a Java Enterprise Edition (JEE) egy szabványa. MVC programtervezési mintát valósít meg, illetve azon alapul.

„Lényege, hogy ne egy class-ba írjunk mindent, hanem szétbontsuk az alkalmazásunkat több rétegre. Model-re, ami az az adat lesz, amivel dolgozni szeretnénk, a Controller-re, ami bizonyos műveleteket hajt végre az adatokon és a View-ra, ami megjeleníti az adatot (…). Léteznek JSF-et kibővítő, különböző cégek által készített JSF komponensek, amik együtt használhatók a standard JSF-fel. Ilyen például a PrimeFaces is". (Horváth, é.n.) Az előbb említett PrimeFaces szolgáltatást én is igénybe vettem.

A keretrendszer View template-nek vagy Faceletnek nevezett XML fájlokat használ a megjelenítési modell leírására. A kéréseket a FacesServlet dolgozza fel, ami azután betölti a megfelelő mintákat, majd kezeli az eseményeket, és létrehozza a választ a kliensnek. Ez többnyire XHTML formátumban történik. A felhasználói felület komponenseit minden lekérés végén elmenti, majd ugyanazon View következő előállításakor újra betölti. (JavaWorld, 2018)

5. XHTML

A felhasználói felületet XHTML (EXtensible HyperText Markup Language) nyelven írtam meg. A nyelv alapja az XML, mely miatt jóval szigorúbb formátuma van, mint a HTML nyelvnek. Emellett viszont sokkal több lehetőségem van ezzel a nyelvvel. Például Java nyelven megírni a köztes osztályokat és vezérelni a honlapon megjelenő objektumokat. Ezen kívül szorosan kapcsolódik a 4. alfejezetben említett Java ServerFaces technológiához. (JavaWorld, 2018)

6. Fejlesztői környezet

Fejlesztésre a IntelliJ IDEA 2018.3.5 Ultimate Edition-t használtam, amelyet tanulói licensszel vettem igénybe.

7. Felhő szolgáltatás

Heroku egy ingyenesen is igénybevető felhőszolgáltatás, mely segítségével a munkára használt gépünkön létrehozott és működő projektjeinket tudjuk az interneten is elérhetővé tenni. (Salesforce.com, é. n.)

A szolgáltatás a Git verziókezelőszoftvert is igénybe veszi a működéshez, így ezt is használtam a projektemhez.

A Git egy nyílt forráskódú, elosztott verziókezelő szoftver, amely a sebességre helyezi a hangsúlyt. (Software Freedom Conservancy, é. n.)

8. Egyéb technológiák

Minimális Javascript, CSS és Perl nyelv is megtalálható a programkódban. A Javascript a weblapon lévő csetfelületet kezeli, a CSS a weblap stílusát, kinézetét hivatott beállítani, míg a Perl a Rivescript és Java közötti tolmács program megírására használt nyelv.

IV. Eredmények

A dolgozat legfőbb eredménye az elkészült chatbot, mely a terveknek megfelelően minimálisan képes beszélgetni a felhasználóval és a keresett kulcsszavak alapján képes keresni a kar honlapjáról elmentett adatok között. Így egy keresőszolgáltatásnak megfelelő funkcióval rendelkezik. Ezen kívül a felhasználónak lehetősége van visszajelezni. A felsorolt válaszok közül felértékelni a megfelelőeket és saját javaslatot is tenni, ha esetleg nem találja meg a megfelelő kulcsszóra a megfelelő linket.
E program felépítését az alábbi ábrán látható. (10. ábra)
A kész program a https://bpg-ttkbot.herokuapp.com/chatbot.xhtml linken található.

1. A program működése

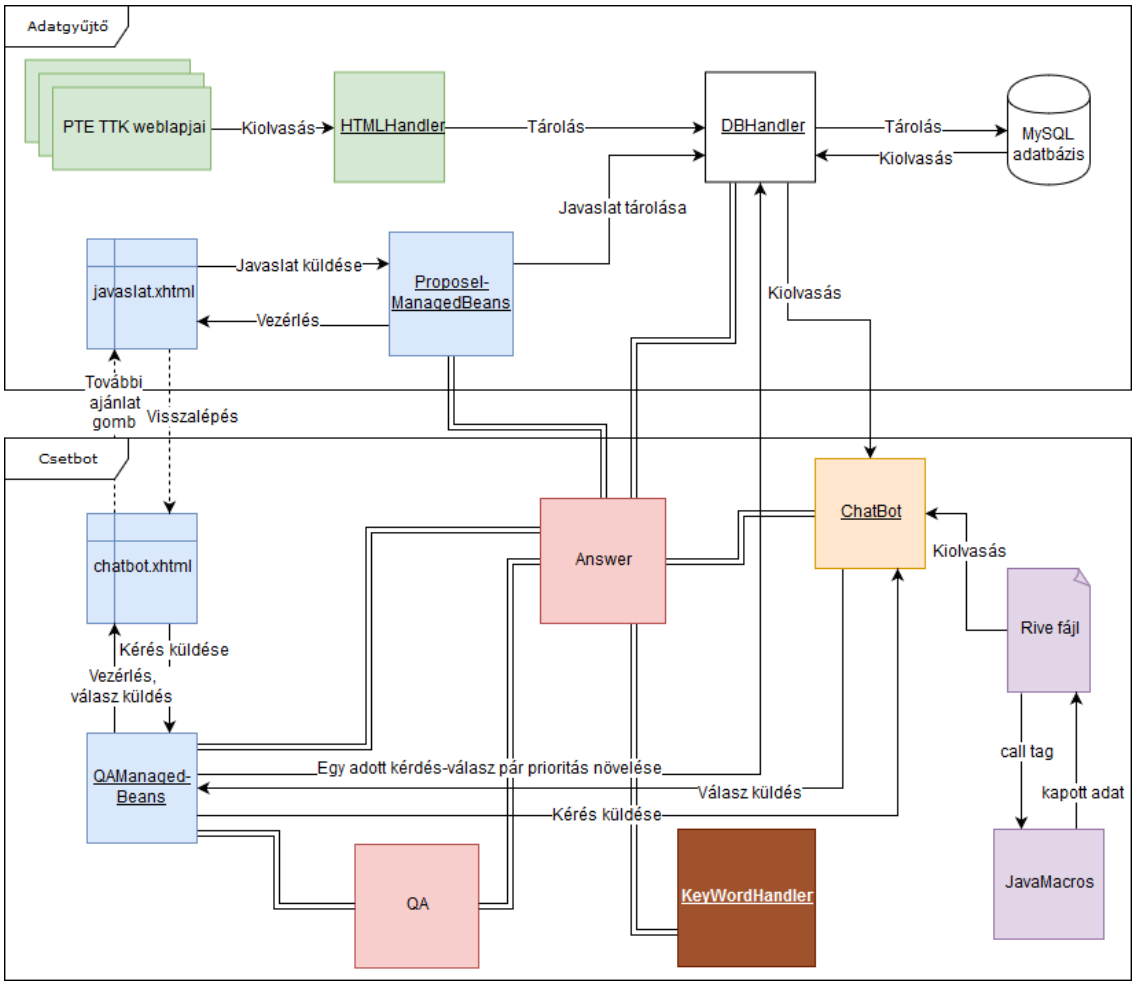
A programot két fő részre bontottam. Ezek nem teljesen független programrészek, de az egyszerűség miatt így utalok majd rájuk.

- **Az adatgyűjtő**

  Feladata a megadott, jelen esetben a PTE Természettudományi Kar honlapján lévő adatok kiolvasása, rendezése és elmentése az adatbázisba. Ehhez tartozik még a felhasználó által beírt javaslatok és az adott válasz felértékelésének kezelése is.

- **A csetbot**

  Feladata a felhasználóval való kommunikáció, kéréseire a megfelelő adatok kiolvasása az adatbázisból és ezen adatok kiírása a felhasználói felületre.



10. ábra: A program működése

Az ábrát a draw.io szolgáltatással készítettem.[9]

A különböző színek az összetevők témáját hivatottak megkülönböztetni. Azonos színű egységek, azonos feladatkörhöz tartoznak. A zöld a kiolvasásra szánt weblapok és az adatok kiolvasásáért felelős HTMLHandler osztály. A fehér az adatbázis és az azt kezelő osztály. A világoskék a két weblap és az őket vezérlő managed beans, vagyis vezérlő osztályok. A rózsaszín az adat típusú osztályokat jelöli. A narancssárga, vagyis a Csetbot class segítségével a felhasználó által küldött kérdést tudja a program visszaadni. A lila összetevők a csetbot „agya”, értelme. Végül a barna, vagyis a KeyWordHandler osztály egy absztrakt osztály, ami rengeteg különböző függvényt tartalmaz, mely megkönnyítette a munkámat.

A sima nyilak az adatok irányát mutatják. A szaggatott nyilak azt, hogy az adott nyílra írt gomb, mely weboldalra viszi a felhasználót. A kettős vonalú, fejnélküli nyíl pedig azt mutatja, hogy egy adat típusú osztály, mely osztályoknak biztosítja az általa reprezentált adatstruktúrát.

[9] https://www.draw.io/

A sima nyilakra írt szöveg, a feladat, ami miatt az adatok mozgatva vannak. Azon osztályoknak a nevét pedig aláhúzással jelöltem, amelyek közvetlenül használják a KeyWordHandler valamely függvényét vagy metódusát.

2. Az adatbázis felépítése

Az adatbázis csupán egy táblából áll. A qa nevű táblából, aminek az oszlopai:

2.1. id

Egy INTEGER típusú mező, amely automatikusan növekszik eggyel, ha új rekord kerül az adatbázisba.

2.2. question

Ide menti a program az oldalon talált szöveget TEXT típusú adatként. Külön kezelem azon szövegeket, amik egy linkhez csatoltak és azt, amihez nem. Típusnak azért nem VARCHAR vagy CHARACTER típust használok, mert nem tudom meghatározni előre, hogy mekkorák lesznek a kiolvasott oldalon lévő szövegek hossza. Átlagban ötezer karakter, de mondjuk a hírarchívum oldalakon több, mint százezer karakter található.
A TEXT típusnak viszont a maximális hossza korlátlan. (The PostgreSQL Global Development Group, é. n.)

2.3. answer

TEXT típusú mező. A szöveghez tartozó vagy a szöveget tartalmazó linket mentettem ide. Ezt az adatok kapja meg a felhasználó válaszként ha a csetbot keresési funkcióját használja.

2.4. priority

INTEGER-ként elmentett adat. Az adott válasz prioritását tárolja. A prioritás kiosztását később részletezem.

3. Első ötlet

Először ki szerettem volna hagyni az adatbázist a program fejlesztésekor, és csak a Rivescript-re hagyatkozva megoldani a problémát. A megoldások tesztelésénél viszont az adatbázisban a keresés gyorsasága jóval felülmúlta a Rivescript-ben való keresést. Emellett beleütköztem a magyar nyelv miatt egy másik problémába is.

A Rivescript-es megoldás az lett volna, hogy egyenként mentem el a talált szavakat a honlapról, amely szavakhoz elmentem azon linkeket, ahol találtam az adott kifejezést. A szavakról egy külső szótár felhasználásával levágtam volna a toldalékot, így megkapva a szótövet. Persze egy szó szerepelhet több weboldalon is, így létrehoztam egy `<send>` tag-et a Rivescripten belül, amivel automatikusan felismerve a szövegben, a csetbot külön-külön küldte volna el szövegeket. Ezt így kellett volna megoldjam, mert ha csak külön válaszokként kezelem ezeket, nem az összeset küldi el, hanem véletlenszerűen egyet. Így nézett volna ki egy mentés:

```
+ alma
- https://www.alma.hu<send>https://www.alma.ttk.hu<send>...
```

13. forráskód: Első terv

A felhasználó által beírt szöveg szavaira aztán külön-külön lefuttattam volna egy keresést a rive fájlban, amelyben az összes, a honlapon szereplő szó és hozzá tartozó link szerepelt volna.

Első nehézség, amibe ütköztem, hogy a szótő keresése egy adott szónak nagyon nehéz feladatnak bizonyult. Találtam egy szótárat[10], amelyet helyesírás ellenőrzésre használnak és a magyar nyelv minden lehetséges toldalékkombinációját tartalmazza. Arra gondoltam, hogy egyezéseket keresve levágom a szavak végéről a toldalékokat.

Ezt gyorsan el kellett vessem, hiszen a magyar nyelvben a szavak végi utolsó betű gyakran változik egy toldalék hatására. Például egy *anyával* szóból levágva a *-val* toldalékot, marad az *anyá*(sic!). Miközben a szótő az anya. Vagy betűcserék is lehetségesek, az úgynevezett tőhangváltások (Pl.: terem – termett). Ezeken kívül is sok más olyan példa létezik, amikor a toldalék megváltoztatja a szótövet. (Magyar Tudományos Akadémia, 2015)

Ezért, a szótő keresésre nem tűnt ez egy működő megoldásnak. Mielőtt viszont a további megoldásokon gondolkoztam volna, összehasonlítottam az adatbázis és a Rivescript keresési gyorsaságát. És ezzel eldőlt melyik technológiát választom.

A Rivescript-es megoldás esetében 100.000 db, maximum 20 véletlen karakterből álló szót generáltam és hozzá egy választ. A szavak végére hozzáfűztem még a sorszámukat. 100-szor lefuttattam a keresést az első, a középső és az utolsó adatra. Végül átlagosan körülbelül 3000 ms volt a keresési idő. Így nézett ki a rive fájl:

[10] ftp://ftp.gnu.org/gnu/aspell/dict/hu/aspell6-hu-0.99.4.2-0.tar.bz2

```
+ zgnéhkácöbqíx39810
- válasz39810

+ kbnüyívtrxsótmoái39811
- válasz39811

+ öujörxwőlúlygíultóahuqö39812
- válasz39812

+ orízeéhákeéúőmnw39813
- válasz39813

+ úepáqpquvdlqőzx39814
- válasz39814
```

11. ábra: Rivescript gyorsaság teszt

Az adatbázisban keresés egy másik mechanikával történik. Az egyik oszlop a *kérdés*, mely az adott honlapon megtalálható összes szöveg. A másik a *válasz*, amely a szöveghez tartozó link. A *kérdés* oszlopban keresek mintákat és hasonlítom össze a felhasználó által beírt szöveggel.

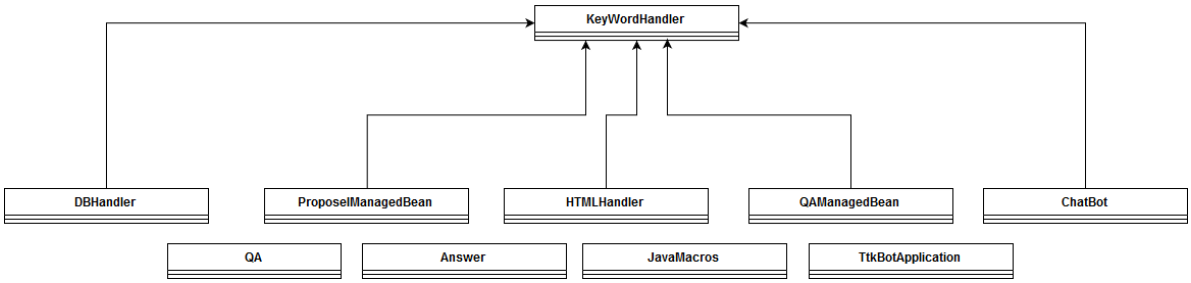
Az adatbázis esetében a teszt a következőkből állt: 1000 darab 10.000 random karakterből álló sort tettem az adatbázisba. A szóközök esélye a random karakterválasztásnál háromszoros volt, ezzel szimulálva a szavakat. Minden 100. sorba manuálisan beleírtam az *alma* szót és rákerestem erre. A keresés gyorsasága 141 ms volt! Így gyorsan eldőlt melyik technológiával fogok dolgozni.

A Rivescript teszt fájlja a test mappában található.[11]

4. Osztályok bemutatása

A program osztály szerkezetét UML osztálydiagrammon keresztül kívánom bemutatni. A diagramm felépítésének módszerét a forrásként megjelölt műből vettem. (Kilián, 2018)

A túl sok osztály és attribútum miatt viszont sajnos nem fért rá egy diagrammra az egész, így külön mutatom be a leszármazásokat és az osztályok felépítését.



12. ábra: Leszármazások

[11] TTKBot\src\main\resources\test\timeTest.rive

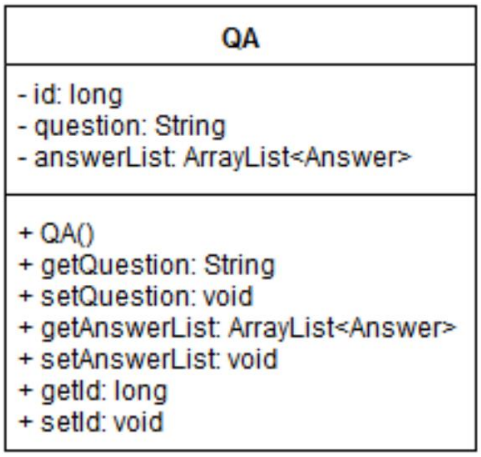
4.1. QA osztály

Adat definiálásra használt osztály. A QA a *question* és az *answer* szavak rövidítései. Ilyen adatstruktúrából álló listában mentem el az adott kérdés-válasz párt, amit aztán kiírok a felhasználónak a csetbot felületén. A kérdés a felhasználói felületről jön, míg a válasz a kérdéstől függően vagy a csetbot agyából, a Rivescript fájlból, vagy az adatbázisból, ahol a weblapból kiolvasott adatokat tárolom.

Attribútumai:

- `private long id`
  - Az adott kérdés-válasz pár egyedi azonosítója.
- `private String question`
  - A felhasználó által beírt szöveg. Azért kérdésként utalok rá, mivel a felhasználó, ha nem is nyelvtani értelemben, kérdést tesz fel, amire egy választ vár rá.
- `private ArrayList<Answer> answerList`
  - A válaszokat tartalmazó lista. Hiszen egy kérdéshez, kéréshez több választ is adhat a csetbot. Az elemek típusa Answer.

A metódusok között a szokásos getterek és setterek találhatók.



13. ábra: QA osztályszerkezete

4.2. Answer osztály

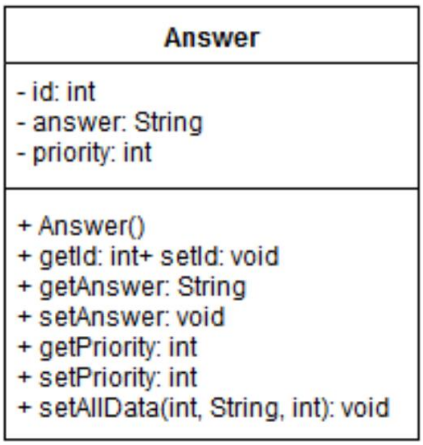
A QA adatstruktúrában található answerList elemeinek típusa. Először csak String típussal szerettem volna dolgozni, de rájöttem, hogy az adatbázisból kiolvasott válasz prioritását és id-jét is használnom kell majd.

Attribútumai:

- `private int id`
  - Az adatbázisból az adott válaszhoz tartozó egyedi azonosító.

- `private String answer`
  - A válasz szövege.
- `private int priority`
  - A válaszhoz tartozó prioritási érték.

A metódusok között a szokásos getterek és setterek találhatók. A `setAllData` nevű metódus pedig egy setter, amely az összes attribútumot képes beállítani.



14. ábra: Answer osztályszerkezete

4.3. KeyWordHandler osztály

Ez egy absztrakt osztály, vagyis implementálni nem lehet, csak leszármaztatni belőle. Egy gyűjtő osztály, ami nem csak a nevében jelzett kulcsszavakkal dolgozik, hanem egyedi String és lista műveletekkel is rendelkezik.

A szótárhoz használt nonKeyWords.txt fájl a BME Média Oktató és Kutató Központjának egyik kutatási projektjén alapul. „A nyelv mentális reprezentációja” c. OTKA Tudományos iskola pályázat keretében készült a projekt. (Balogh, Halácsy, & Szalai, 2014.)

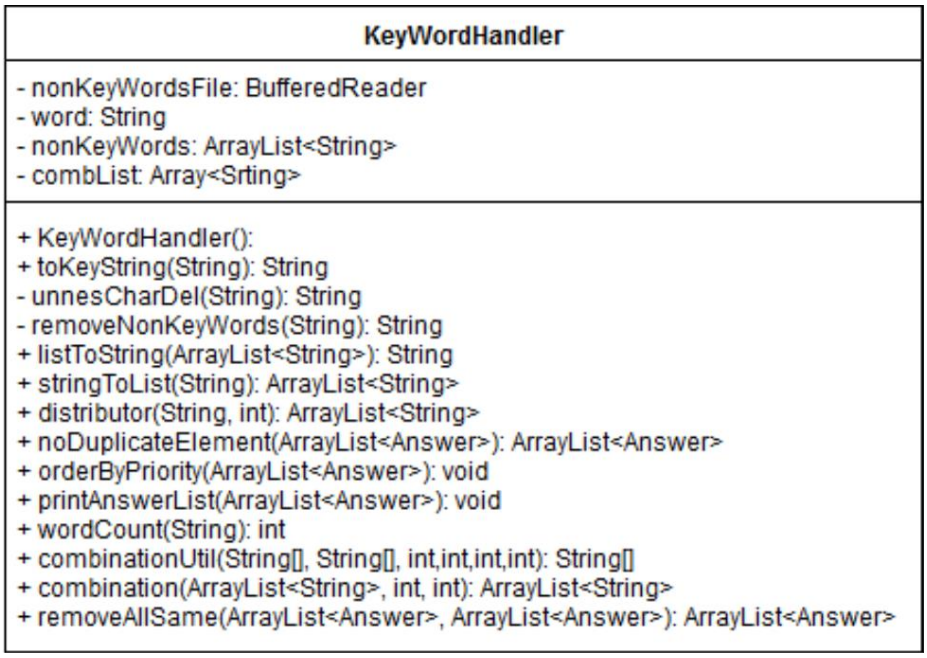
Ez egy magyar szavakat tartalmazó szótár. A txt fájlból kitöröltem azon szavakat, amiket kulcsszavaknak tulajdonítottam: a fő-, mellék- és számneveket és természetesen az igéket. Ezt Microsoft Excel táblázatkezelő segítségével tettem meg. Így a szótár a nem kulcsszavakat tartalmazza.

Függvények/metódusok:

- `public KeyWordHandler()`
  - Az osztály konstruktora, ami megnyitja a nonKeyWords.txt szövegfájlt, és egy listában tárolja el azokat úgy, hogy a nagybetűket kicsivé alakítja.
- `public String toKeyString(String string)`
  - A megadott String-ben felcseréli a nagybetűket kisbetűkké, és ezen kívül az `unnesCharDel` és a `removeNonKeyWords` függvényeket is lefuttatja.

- `private String unnesCharDel(String string)`
  - A megadott karakterláncból eltávolítja a megadott karaktereket.
- `private String removeNonKeyWords(String string)`
  - A megadott karakterláncból kitörli a nem kulcsszavakat a nonKeyWords.txt szótár alapján.
- `public String listToString(ArrayList<String> list)`
  - Egy szavakból álló listát String-é alakít, amelyben a szavak szóközökkel vannak elválasztva egymástól.
- `public ArrayList<String> stringToList(String string)`
  - Egy karakterláncban a szóközökkel elválasztott szavakat teszi egy ArrayList-be, ahol a lista egy eleme a string egy szava.
- `public ArrayList<String> distributor(String string, int level)`
  - A megadott string-et level db szóra vágja fel a függvény, de csakis az egymás melletti szavak mentén. Pl.: string = ”alma körte répa cékla”, level = 2, a lista:
    - alma körte
    - körte répa
    - répa cékla
- `public ArrayList<Answer> noDuplicateElement(ArrayList<Answer>)`
  - Egy Answer típusú objektumokból álló listából kiválogatja azon elemeket, ahol az `answer` vagyis a válasz ugyanaz, és csak egyet hagy a listában ezek közül.
- `public void orderByPriority(ArrayList<Answer> answerList)`
  - A megadott Answer típusú objektumokból álló listában sorba teszi az elemeket a `priority` szerint növekvő sorrendben.
- `public ArrayList<String> combination(ArrayList<String> arl, int n, int r)`
  - A bemenő lista szavain végez ismétlés nélküli kombinációt a megadott paraméterek alapján. Az `n` az összes szó darabszáma, az `r` pedig, hogy mennyit szeretnénk kiválasztani az összesből. Más szóval `n` alatt az `r` darab ismétlés nélküli kombinációt készít a függvény a megadott szavakból. A függvény a hivatkozásban megjelölt oldalon található függvény alapján készült. (GeeksforGeeks, é. n.)
- `public String[] combinationUtil(String arr[], String data[], int start, int end, int index, int r)`
  - A `combination` függvény segédfüggvénye. (GeeksforGeeks, é. n.)
- `public void printAnswerList(ArrayList<Answer> answerList)`
  - Egy Answer típusú lista elemeinek adatait írja ki a képernyőre.

- `public int wordCount(String string)`
  - A megadott karakterláncban található szavak számát adja meg.
- `public ArrayList<Answer> removeAllSame(ArrayList<Answer> a, ArrayList<Answer> b)`
  - Ezt egy halmazműveletként lehet értelmezni. Az „a” nevű halmazból (listából) ki lesz vonva a „b” nevű halmaz (lista).



15. ábra: KeyWordhandler osztályszerkezete

4.4. DBHandler osztály

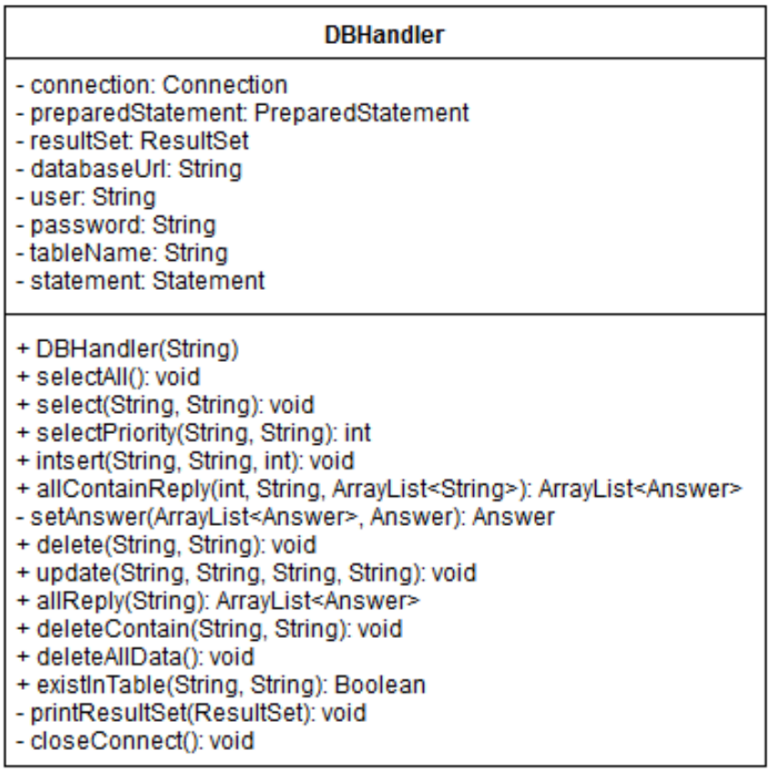
Ezen osztály segítségével kezelem a Java-n belül az adatbázist.

Metódusok:

- `public DBHandler(String tableName)`
  - Csatlakozik a paraméterekben megadott jelszóval és felhasználónévvel az adatbázishoz és beállítja a tábla nevét, amin ezek után dolgozni fog.
- `public void selectAll()`
  - Ez a metódus egy `SELECT * FROM tableName` parancsnak felel meg. Az eredményeket kiírja a képernyőre.
- `public void select(String columnName, String param)`
  - Ez a metódus egy `SELECT * FROM tableName WHERE columName = param` parancsnak felel meg. Az eredményeket kiírja a képernyőre.
- `public int selectPriority(String columnName, String param)`
  - Ezen függvény egy megadott rekord priority értékét adja vissza.

- `public void insert(String question, String answer, int priority)`
  - A metódus egy `INSERT` parancsnak felel meg, ahol minden adatot megadunk az `id` kivételével, ami automatikusan növekszik.
- `public ArrayList<Answer> allContainReply (int level, String columnName, ArrayList<String> args)`
  - A metódussal a bemenő lista szavainak bizonyos kombinációja alapján készítek egy lekérést. A szavak száma mínusz a level értéke számmal lefuttatom a KeyWordHandler combination metódusát. Majd a visszakapott értékből készítem a lekérdezést.
  - Pl.: Az input: „alma körte banán, citrom”. Ha level=1, meghívom a `combination(„alma, körte, banán, citrom”, 4, 3)` függvényt a megadott paraméterekkel. Majd hozzáfűzök az eredményhez bizonyos szavakat, hogy egy lekérdezés parancsot kapjak.
    - `SELECT * FROM tableName WHERE columnName LIKE '%alma%' and columnName LIKE '%körte%' and columnName LIKE '%banán%' or columnName LIKE '%alma%' and columnName LIKE '%körte%' and columnName LIKE '%citrom%' or columnName LIKE '%alma%' and columnName LIKE '%banán%' and columnName LIKE '%citrom%' or columnName LIKE '%körte%' and columnName LIKE '%banán%' and columnName LIKE '%citrom%'`
  - Majd a parancsot végrehajtja a függvény.
  - A ’%’ karakter állítottam be, hogy ne csak az egész szavakat keressen a szövegben.
- `private Answer setAnswer(ArrayList<Answer> answerList, Answer, answer)`
  - A keresési eredményeket menti el egy Answer típusú listában.
- `public void delete(String columnName, String param)`
  - A metódus egy `DELETE FROM tableName WHERE columnName = param` parancsnak felel meg.
- `public void update(String upColumnName, String upParam, String weColumnName, String weparam)`
  - A metódus egy `UPDATE tableName SET upColumnName = upParam WHERE weColumnName = weparam` parancsnak felel meg.
- `public ArrayList<Answer> allReply(String question)`
  - Ezzel a metódussal minden olyan sort ki tudok olvasni az adatbázisból, amelyek tartalmazzák a *question* nevű stringet.

- `private void deleteContain(String columname, String contain)`
  - A metódus azon sorokat törli ki az adattáblából, amelyek tartalmazzák a *contain* nevű stringet.
- `public void deleteAllData()`
  - Minden adatot töröl az adattáblából. Az id előről kezdi a növekedést.
- `public Boolean existInTable(String question, String answer)`
  - A függvény azt figyeli, hogy a megadott *question* és *answer* pár szerepel-e már a táblázatban.
- `private void printResultSet(ResultSet rs)`
  - A megadott resultSet, vagyis egy adott parancs után kinyert adatokat írja a képernyőre.
- `private void closeConnect()`
  - Megszűnteti a kapcsolatot az adott objektum és az adatbázis között. A @PreDestroy annotáció segítségével érem el, hogy a metódus az objektum megszűnése előtt fusson le minden esetben.



16. ábra: DBHandler osztályszerkezete

4.5. HTMLHandler osztály

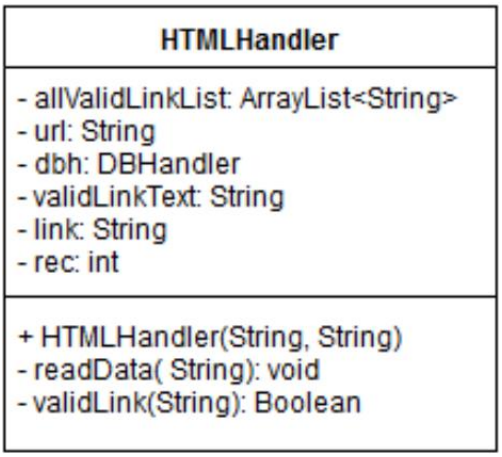
Az adatok kiolvasásáért felelős osztály. Feladata minden megadott weblapról kiolvasni az adatokat. Normál esetben, ha lenne lehetőségem arra, hogy egy szabványos csatornán automatikus értesítéseket kaphassak a honlapon bekövetkezett minden változásról, elég lenne a honlap frissítése után lefuttatni egy példányt, de, mivel nincs erre lehetőségem, ütemezve minden hétfőn 1:30 perckor fut le az algoritmus és olvassa ki a megfelelő adatokat.

Metódusok:

- `public HTMLHandler(String url, String tableName)`
  - Már az objektum létrehozásával lefut az összes szükséges metódus. A weboldalról kiolvassa az összes kért adatot és frissíti a táblát, ha esetleg új adat keletkezett volna. A frissítésnél első körben megvizsgálom, hogy létezik-e az adott kérdés-válasz pár és ha létezik, akkor azt is vizsgálom, hogy adott `answer`-hez tartozó `question` szövege nagyságrendileg megegyezik-e azzal, amit el akarok menteni. Mert ha igen, valószínűleg hozzáírtak az előző mentés óta. Ebben az esetben felülírom a meglévő adat `answer` mezőjét. Egyéb esetben külön adatként hozzáadom a táblához.
  - Ebben a feladatban nagyon sokat segített a Jsoup Java könyvtára. Nagyon kényelmes API-t biztosított az adatok kinyeréséhez és manipulálásához. (Hedley, 2018.)
- `private void readData(String url)`
  - Ezen metódus segítségével olvassa ki a program az adott URL-ből a megadott adatokat. Az adatokat a következőképpen tárolom:
    - Ha egy szöveghez tartozik link, vagyis a HTML kódban az `<a>` tag-en belül van a szöveg, a *href* utáni linket az *answer* oszlopba mentem, míg a hozzá tartozó, a honlapon megjelenő szöveget a *question* oszlopba. Ennek a prioritása 2. Minél nagyobb egy válasz prioritása, annál előrébb sorolom, mikor kilistázásra kerül a felhasználónak. Ebben az esetben kivételek azok a linkek, amikben megtalálható a hírek vagy a hírarchívum szó. Azoknak a prioritása 1.
    - A weboldalon található szimpla szöveg a *question* oszlopba, míg a szöveget tartalmazó weboldal linkje az *answer* oszlopba kerül. Előtte viszont a szövegből kitörlöm azon szövegrészleteket, amik az első pontban szerepeltek, vagyis amikhez tartozik link. Ezeknek 1 prioritás értéket adtam.
    - A felhasználó által beírt javaslat a legkevesebb prioritású, vagyis 0, de ezt a műveletet a `QAManagedBean plusPriority` metódusa kezeli.

        - Ezen metódus rekurzívan lefut azon linkekre, amik megfelelnek a validLink függvény által előállított feltételeknek. Ha nem felel meg, például egy kép esetében, a kép címét és a kép URL-jét elmenti, de nem próbálja lefuttatni ezt a metódust a kép URL-jére.

- `private Boolean validLink(String link)`
    - Ez a függvény dönti el azt, hogy az adott linkre le kell-e futtatni a readData metódust. A célom csak a *ttk.pte.hu* körzethez tartozó oldalak kiszűrése volt. Bár rájöttem, hogy az adott URL-hez sok egyéb aloldal is tartozik, mint például a *joido.ttk.pte.hu* vagy az *szjszk.ttk.pte.hu*. Mivel ezek között nem volt célom a program bemutatása, ezeket is kiszűrtem. Ugyanúgy, ahogy az angol oldalakat. Természetesen a feltételek átállításával az ezeken megtalálható adatokat is le lehetne menteni.



17. ábra: HTMLHandler osztályszerkezete

4.6. Chatbot osztály

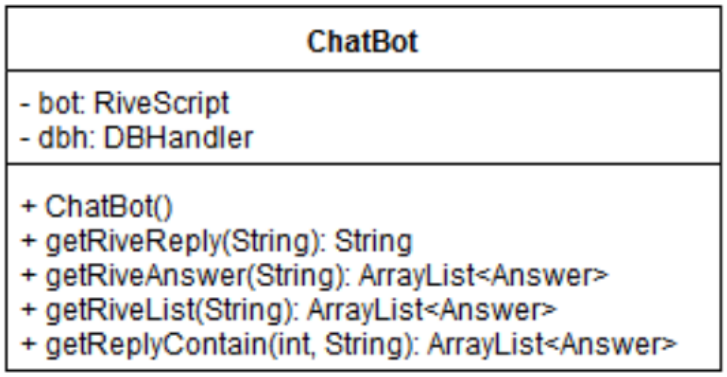
Ezen osztály biztosítja a kérdésekre adott válaszokat. Három féle módon is lekérdezhető egy adott kérdésre a válasz.

Metódusok:

- `ChatBot()`
    - Az osztály konstruktora. Megnyitja a megadott .rive kiterjesztésű fájlt. Jelen esetben a *ttkBot.rive*-ot. Beállítja a karakterkódolást és a Perl nyelven megírt interpreter fájlt is megnyitja.
    - Ennek a neve: *rsp4j.pl*. A fájlt teljes mértékben a hivatkozásban lévő programból használtam fel. (Petherbridge & Overdijk, 2016)

- A fájlokat a felhőrendszerre telepítés miatt, nem tudtam egyszerűen csak megnyitni. Adatfolyamba, úgynevezett stream-be kellett töltsem az adatokat és onnan tudtam használni a fájlban található információt. Az itt található forráskód egy részét a forrásban megjelölt weboldalról használtam fel. (HowToDoInjava.com, 2018)

- `public String getRiveReply(String msg)`
  - Egy string üzenetre visszaad egy string választ.
- `public ArrayList<Answer> getRiveAnswer(String msg)`
  - Egy string üzenetre visszaad egy egy elemű listát, amelybe a rive-ból kiolvasott válasz Answer típusként kerül elmentésre.
- `public ArrayList<Answer> getReplyList(String msg)`
  - Az adatbázisból kiolvassa a válaszokat a `DBHandler allReply` nevű függvénye segítségével.
- `public ArrayList<Answer> getReplyContain(int level, String msg)`
  - Az adatbázisból kiolvassa az válaszokat a `DBHandler allContainReply` nevű függvénye segítségével.



18. ábra: Chatbot osztályszerkezete

4.7. JavaMacros osztály

Ebbe az osztályba került a Rivescript próba makrója. A programrészletet teljes mértékben a hivatkozásban lévő forrásból vettem át. (Petherbridge & Overdijk, 2016)

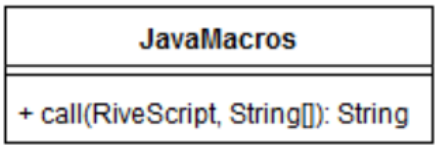
Metódusok:

- `public String call (com.Rivescript.Rivescript rs, String[] args)`
  - A Rivescriptben call tag-ként meghívandó metódus, amely így néz ki:

```
+ mondd visszafele *
- <call>javatest <star></call>
```

14. forráskód: Makró bemutatása

- A függvény lényege, hogy a beírt „mondd visszafele” kifejezés után a szót vagy szavakat visszafele írja ki a felhasználónak. Ez azt bizonyítja, hogy a szavakat és válaszokat a Rivescript-en belül is lehet manipulálni egy külső nyelv segítségével.



19. ábra: JavaMacros osztályszerkezete

4.8. ManagedBeans osztályok

A Managedbean osztály a Java ServerFaces speciális osztályai, amelyekkel vezérelhetjük az XHTML nyelven megírt weblapunkat. Ezek a kontroller osztályok. A ManagedBeans osztályokban kötelező, úgynevezett annotációkat, getterek és settereket használni, mert csak ezeken keresztül tud az osztály kommunikálni a felülettel. Az annotáció egy olyan nyelvi eszköz, mely plusz információt hordoz a fordító számára. (Pl.: @ManagedBean annotáció segítségével tudjuk megjeleníteni a felületen az adatainkat, vagy a @Scheduled annotációkat tudunk ütemezni feladatokat stb.) (Horváth, é.n.)

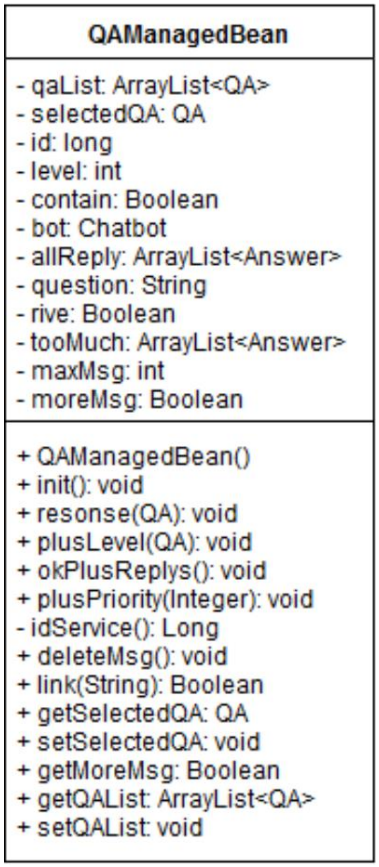
4.9. QAManagedBeans osztály

Az osztály az chatbot.xhtml felület vezérlésére lett létrehozva.

Metódusok:

- `public QAManagedBean()`
  - Az osztály konstruktora. Alaphelyzetbe állítja a változókat.
- `public void init()`
  - Az inicializálásnál lefutó metódus. Ezt a @PostConstruct annotációval tudjuk elérni. Itt adható meg, mi lesz a kezdőüzenet.
- `public void response(QA qa)`
  - A felhasználó a 'Kérdés...' inputba beírt szövegére próbál a metódus választ adni. A Chatbot osztály egy példánya segítségével először a rive fájlban keres, majd az adatbázisban, de ott is először a getReplyList metódus segítségével a szavak elhelyezkedésének figyelembevételével. Majd, ha nincs találat, a figyelembevétele nélkül, a getReplyContain metódus segítségével. Majd, ha akkor sincs találat, automatikusan meghívja a plusLevel metódust. A teljes keresési algoritmusra a felsorolás végén térek ki.

- `public void plusLevel(QA qa)`
  - A 'További találatok' gomb megnyomásával aktiváljuk ezt a metódust. Minden gombnyomásra egy újabb szinttel mélyebben keres. A már kiadott találatokat nem mutatja meg újra.
- `public void okPlusReplys()`
  - Ez a metódus a 'További találatok' gomb megjelenéséért felelős. Ha nincs több szint, lehetőség a további válaszok találatára, nem engedi megjelenni a gombot.
- `public void plusPriority(Integer id)`
  - Minden nem Rivescriptből kapott válasz esetén lehetőségünk van egy válasznak növelni a prioritását. Ha megnyomtuk a gombot ez a függvény fut le és a megfelelő `id` segítségével növeli az adatbázisban a megfelelő rekord `priority` értékét.
- `private long idService()`
  - A QA listában `id`-t növelő metódus.
- `public void deleteMsg()`
  - Az 'Üzenetek törlése' gomb megnyomására lefutó metódus. Kiüríti a megjelenítendő kérdés-válasz listát és alaphelyzetbe állítja a változókat.
- `public Boolean link(String string)`
  - A válaszokban, a link felismeréséért felelős függvény.
- `public ArrayList<QA> getQAList()`
  - a QAList változót adja vissza.
- `public void setQAList(ArrayList<QA> qaList)`
  - a qaList változónkat állíthatjuk be ezzel.
- `public QA getSelectedQA()`
  - a selectedQA változónkat adja vissza.
- `public void setSelectedQA(QA selectedQA)`
  - a selectedQA változót tudja beállítani.
- `public Boolean getMoreMsg()`
  - Visszaadja a moreMsg értékét, ami 'További találatok' megjelenítéséért felel.



20. ábra: QAManagedBean osztályszerkezete

A keresési algoritmus:

A cél, a beírt kulcsszavak alapján a megfelelőrekordok megtalálása és a találatok fontossági sorrendjének beállítása.

A program írásánál tudtam, hogy létre kell hozzak valamilyen keresési algoritmust. Nem elég az a feltétel, ha a felhasználó által beírt szavak szerepelnek az adatbázis *question* oszlopának valamely mezőjében, hiszen a felhasználó írhat több szót, amik közül nem feltétlen szerepel mind a szövegben. Ezen kívül számolnom kellett azzal is, hogy a kulcsszavakhoz toldalék is kapcsolódhat. A harmadik fontos dolog pedig az volt, hogy bizonyos szempontok alapján priorizálnom kell a megtalált válaszokat, hiszen egy-egy kulcsszó akár szerepelhet több száz weboldalon is, de csak 1-2 lesz, ami igazán fontos.

Erre sajnos nem találtam olyan forrást, amely pontosan megmutatott volna egy jó, működő megoldást. A Google esetében megfigyelt tapasztalatok alapján viszont ki tudtam indulni.

A megfigyelésem szerint, ha beírunk bizonyos szavakat a keresőbe, a találati listában olyan találatok is szerepelnek, amelyekben nem feltétlen található meg az összes beírt kulcsszó. Viszont azokat sorolja előrébb, amely oldalakban a legtöbb kulcskifejezés fellelhető. Ezen kívül azokat a találatokat is előrébb sorolja, amelyekben a szavak pontosabban megjelennek. Például, ahol a beírt kifejezések sorrendje megfelel a weboldalon találta szavak sorrendjével,

azt fontosabbnak tartja, mint ahol csak elszórva találja meg ezeket.
Tudom ezek csak részinformációk. A Google algoritmusa biztosan sokkal több információt vesz figyelembe egy keresésénél. Nekem azonban egyelőre elég volt ennyi.
Így a következőket vettem figyelembe a keresésnél: mennyire pontosan találja meg az adott weboldalon a beírt szavakat a szavak sorrendjét figyelembe véve és hogy mennyi szót talál meg a keresett kifejezésekből.
A weblap kiolvasásánál és a felhasználó által beírt szöveg beolvasásánál is kiszűrtem azon szavakat, amik nem lehetnek kulcsszavak. Pontosabban, a fő-, a mellék-, számneveket és az igéket hagytam csak meg, abból a megfontolásból, hogy ha például a keresett elem az „a és b" és van egy oldal, ahol ez található: „a vagy b", ne legyen kevésbé fontos, mert különbözött a kötőszó. Így csak is a kulcsszavak maradnak meg, a szavak sorrendjének megtartásával.
Legelső lépésben az algoritmus, a Rivescript-ben, a bot agyában keres egyezést. Ha nincs, az azt jelenti, hogy a felhasználó valószínűleg keresni szeretne valamit, nem beszélgetni.
Ezután az algoritmus egyre magasabb szinteken keres, amíg nem talál legalább egy találatot. Minél magasabb szinten van a keresés, annál „engedékenyebb" a program. De ezeken a szinteken belül is két alszintet különböztetek meg. Az első, ahol figyelembe veszi a szavak sorrendjét a keresésnél (`getReplyList` metódus), a második, ahol nem (`getReplyContain` metódus).
Az első esetben, vagy más szóval első alszinten a szint értéke azt adja meg, hogy hány fele bontjuk az adott kulcsszavakat tartalmazó string-et úgy, hogy a szavak sorrendje nem változik. A szétbontás a `KeyWordHandler distributor` függvény segítségével történik, amely következtében, az első szint kivételével, több string-et fogunk kapni. Ezen kifejezésekre külön-külön az algoritmus rákeres és egy listába gyűjti a találatokat. Ha a lista nem üres, kilistázza azt a felhasználónak. Ekkor a felhasználónak van lehetősége további találatokat kérni, ami ebben az esetben nem a következő szintre, hanem a második alszintre viszi az algoritmust. Ha üres a lista, automatikusan megteszi ezt a program.
A második alszinten, a szint értéke azt határozza meg, hogy a beírt kulcsszavakból mennyinek kell kötelezően az adatbázis egy elemében lennie. Az első szinten az összes szónak benne kell lennie az adott dokumentumban, de az egyre magasabb szinteken egyre kevesebbnek. Például, ha a szint értéke elérte a szavak számát, elég, ha csak egy van benne a kulcskifejezésekből az adott dokumentumban. Ezen esetben, ha nincs találat vagy a felhasználó a 'További találatok' gombbal bővíteni szeretné a keresést, a szint értéke megnő egyel.
Azokat az eredményeket, amiket megtalált már a program, nem listázza ki újból, és nem

veszi figyelembe. Ha 100-nál több találata van egy keresésnek, a program nem írja ki a találatokat, hanem küld egy üzenetet a felhasználónak, hogy pontosítson a keresésen.

Példa:

Ha a bemenet ez: *Rébay Viktor és Heisenberger Zsolt telefon*

Kulcsszavak sorrendben: *Rébay Viktor Heisenberger Zsolt telefon*

A keresési algoritmus a 20. ábrán tekinthető meg. A szavak vagy kifejezések dőlt betűvel jelennek meg, a logikai operátorok pedig félkövéren.

A program úgy keres, hogy igaz legyen az adott mezőre a logikai formula. Sorrendben a 2.-at megfigyelve például, ha mindegyik szó megtalálható a szövegben, akkor az egy jó találat. Sorrendben a 3.-at megnézve pedig vagy az egyik, vagy a másik kifejezésnek kell szerepelnie a dokumentumban, akkor kerül a kilistázandó válaszok közé. Az utolsó szinten mindkét alszinten ugyanaz a parancs áll.

Sorrend	szint/alszint	Milyen minta alapján keres?
0.	0.	A Rivescript-ben keres egyezést
1.	1/1	*Rébay Viktor Heisenberger Zsolt telefon*
2.	1/2	*Rébay* **and** *Viktor* **and** *Heisenberger* **and** *Zsolt* **and** *telefon*
3.	2/1	*Rébay Viktor Heisenberger Zsolt* **or** *Viktor Heisenberger Zsolt telefon*
4.	2/2	*Rébay* **and** *Viktor* **and** *Heisenberger* **and** *Zsolt* **or** *Rébay* **and** *Viktor* **and** *Heisenberger* **and** *telefon* **or** *Rébay* **and** *Viktor* **and** *Zsolt* **and** *telefon* **or** *Rébay* **and** *Heisenberger* **and** *Zsolt* **and** *telefon* **or** *Viktor* **and** *Heisenberger* **and** *Zsolt* **and** *telefon*
5.	3/1	*Rébay Viktor Heisenberger* **or** *Viktor Heisenberger Zsolt* **or** *Heisenberger Zsolt telefon* **or**
6.	3/2	*Rébay* **and** *Viktor* **and** *Heisenberger* **or** *Rébay* **and** *Viktor* **and** *Zsolt* **or** *Rébay* **and** *Viktor* **and** *telefon* **or** *Rébay* **and** *Heisenberger* **and** *Zsolt* **or** *Rébay* **and** *Heisenberger* **and** *telefon* **or** *Rébay* **and** *Zsolt* **and** *telefon* **or** *Viktor* **and** *Heisenberger* **and** *Zsolt* **or** *Viktor* **and** *Heisenberger* **and** *telefon* **or** *Viktor* **and** *Zsolt* **and** *telefon* **or** *Heisenberger* **and** *Zsolt* **and** *telefon* **or**
7.	4/1	*Rébay Viktor* **or** *Viktor Heisenberger* **or** *Heisenberger Zsolt* **or** *Zsolt telefon*
8.	4/2	*Rébay* **and** *Viktor* **or** *Rébay* **and** *Heisenberger* **or** *Rébay* **and** *Zsolt* **or** *Rébay* **and** *telefon* **or** *Viktor* **and** *Heisenberger* **or** *Viktor* **and** *Zsolt* **or** *Viktor* **and** *telefon* **or** *Heisenberger* **and** *Zsolt* **or** *Heisenberger* **and** *telefon* **or** *Zsolt* **and** *telefon*
9. = 10.	5/1 = 5/2	*Rébay* **or** *Viktor* **or** *Heisenberger* **or** *Zsolt* **or** *telefon*

20. ábra: Keresési algoritmus

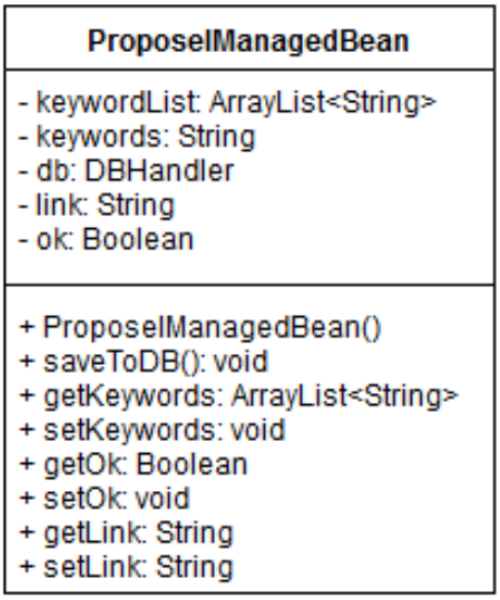
Ezeken után az azonos szinten és alszinten kilistázott találatokat rendezem prioritás szerint csökkenő sorrendbe. A felhasználó előtt mindig a legnagyobb prioritású válasz áll és ő tud keresni a kevésbé fontosabbak között, ha feljebb görget a csetbot ablakban. A prioritást a weblapból olvasásánál határozom meg, több szempont szerint. A prioritások meghatározását a 4.5. fejezetben a `readData` függvény magyarázatakor részletezem. A prioritást a felhasználó képes növelni, ezzel téve hatékonyabbá a keresést.

4.10. ProposelManagedBean osztály

A javaslatok.xhtml oldal vezérlésére létrehozott osztály.

Metódusok:

- `public proposelManagedBean()`
  - Konstruktor. Egy DBHandler példányt hoz létre.
- `public ArrayList<String> getKeyWords()`
  - Visszaadja a beírt kulcsszavak listáját.
- `public void setKeyWords(ArrayList<String> keywordList)`
  - Beállíthatja a kulcsszó listát.
- `public String getLink()`
  - Visszaadja a beírt linket.
- `public void setLink(String link)`
  - Beállítja a linket.
- `public Boolean getOk()`
  - Az `ok` változót adja vissza. Az `ok` változó a hibaüzenet megjelenéséért felel.
- `public void setOk(Boolean ok)`
  - Az `ok` változót állítja be.
- `public void saveToDB()`
  - A ’Mentés’ gomb megnyomása után lefutó metódus. Elmenti a beírt értékeket az adatbázisba.



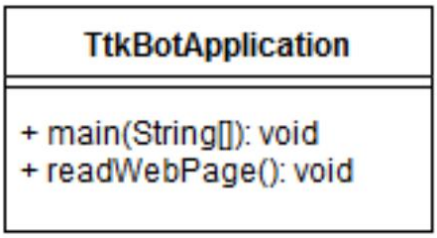
21. ábra: ProposelManagedBean osztályszerkezete

4.11. TtkBotApplication osztály

A main metódussal rendelkező osztály.

Metódusok:

- `public void main(String[] args)`
  - Futtatható metódus. Ez a metódus indítja el a program működését.
- `public void readWebPage()`
  - Ütemezett feladat, melynek az ütemezését a @Scheduled annotáció biztosítja. Minden hétfőn éjjeli 1:30-kor lefut a HTMLHandler és kiolvassa a kar honlapján található adatokat, majd frissíti az adatbázist.



22. ábra: TtkBotApplication osztályszerkezete

5. A felület

5.1. Csetbot felülete

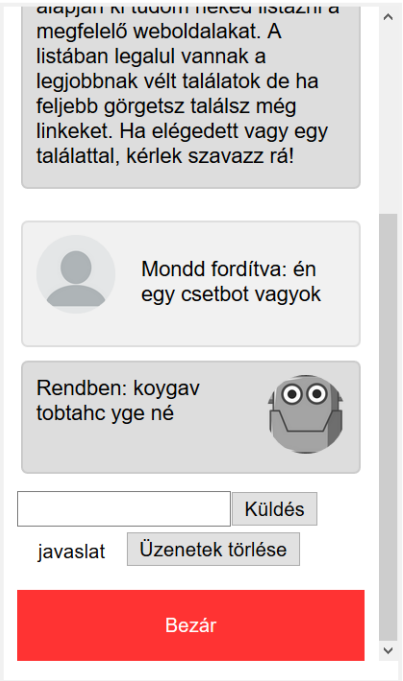
A felület megvalósításánál sokat merítettem a megjelölt forrásokból: (w3schools, 2019), (w3schools, 2019).

Alulról felfele haladva:

A csetbot felülete a jobb alsó sarokban helyezkedik el.

- **Bezár gomb**: Bezárja, lecsukja a csetbot felületet, melyet így a Chat gomb vált fel. Ezzel újra meg lehet nyitni a cset ablakot.
- **Javaslat gomb**: Ezzel a javaslat aloldalt lehet megnyitni.
- **Üzenetek törlése**: Ezzel a csetablakban megjelenő üzeneteket lehet törölni. Ez akkor lehet hasznos, ha nem szeretnénk megnézni az előző felhasználó kereséseit. A kezdő üzenetet nem törölhető.
- **Üzenet input mező**: A felhasználó ide tudja beírni a kívánt üzenetet vagy kulcsszavakat.
- **Küldés gomb**: Elküldi a csetbotnak az üzenetet kiértékelésre, amelynek hatására valamilyen választ vagy válaszokat fog kapni a felhasználó.

- **Üzenet ablak**: A kezdő üzenetet, a kérdések és válaszok találhatók meg itt szürke panelekben. A csetbot üzenete sötétebb szürke panelben, egy robot avatárral[12] a jobb oldalán, míg a felhasználó üzenete világosabb szürke panelben balra igazított ember ikonnal[13].
- **Prioritás növelése gomb**: '^' szimbólummal ellátott gomb az adott üzenethez tartozó rekord prioritását növeli. Ez a gomb csak azon üzeneteknél jelenik meg, amin link található.



23. ábra: Csetbot felülete

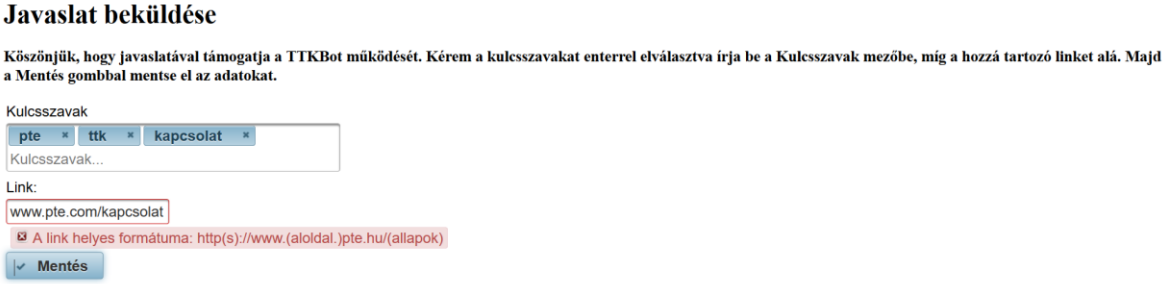
5.2. Javaslat beküldése felülete:

Felhasználóként lehetőségünk van javaslatot küldeni, amelyet a rendszer automatikusan elment az adatbázisba.

- **Legfelül:** a cím és a beküldési instrukciók találhatók.
- **Kulcsszavak input mező**: Ide írhatók a kulcsszavak, melyeket enter megnyomásával választhatunk el egymástól. Üres mező esetén nem engedi elmenteni az adatokat.
- **Link input mező**: Ide írható a link, amely csak a pte.hu valamely aloldalára mutathat, emellett http és https-el is kezdődhet a link, de akár ez ki is hagyható. A link helyes formátuma: (http(s)://)www.(opcionális aloldal.)pte.hu(/opcionális elérési útvonal).

[12] https://www.kisspng.com/png-robot-head-android-clip-art-cartoon-robot-pictures-125510/
[13] https://pixabay.com/vectors/blank-profile-picture-mystery-man-973460/

- **Mentés gomb**: Ezzel menthetjük az adatbázisba a javaslatot.
- **Üzenetek:** Az input mezők alatt jelennek meg az esetleges hibaüzenetek. A sikeres mentést követően, egy felugró ablak tájékoztatja a felhasználót a jobb felső sarokban.



24. ábra: A javaslat oldal kinézete hibaüzenettel

6. Felhő szolgáltatás használata

Az ingyenes Heroku felhőszolgáltatást vettem igénybe a programom internetes elérhetőségéhez. Az ingyenes fiók és adatbázishasználat viszont a program gyorsaságának csökkenésével is járt. Így a keresésekre akár több másodpercet is várhatunk, ami a tesztelések során a helyi hálózaton fél másodpercbe telt.

A szolgáltatás a Git verziókövető rendszerre kapcsolódva használja a programot. Így, ha módosítani szeretném az applikációt, csak a Git rendszerén belül kell ezt megtegyem.

V. Eredmények megvitatása

1. További lehetőségek

Az elkészített funkciókon kívül egyéb lehetőségeket is látok egy ilyen csetbotok felhasználására. Segíthet különböző közvéleménykutatásokban, információ kinyerésben. Esetleg az egyetem iránt érdeklődőknek gyújthat tájékoztatást és pozitív benyomást az intézményünkről, ha arra is beprogramozzuk. De akár egy esemény reklámozására vagy általános információ hirdetésére is használható lenne. Természetesen csak akkor, ha implementálva van a bot a weboldalon.

2. Rivescript gyengesége

Az eszközöket és a nyelvet figyelembe véve azt gondolom, a lehetőségek adottak, hogy a Rivescript nyelven egy intelligens csetbotot írjunk meg, de sajnos nem magyar nyelven.
Ez a nyelv csak is az izoláló, vagyis az elszigetelő nyelvekkel tud igazán hatékonyan működni, mint például az angollal. A magyar agglutináló nyelv, amely azt jelenti, hogy magához ragasztja a toldalékokat és akár a szótő is megváltozhat a toldalékhoz képest. Ezeket a tőhangváltozásokat és toldalékokat a Rivescript nem észleli vagy kezeli. Ezen kívül a magyar nyelv mondatösszeállítása is lazább, mint például az angol nyelvnek, így egy egyszerű mondatot is többféle szórenddel tudunk elmondani, amit pedig a Rivescript-el egyenként le kellene kezelni. (Havas, 1974)
A programban egy nagyon alap szinten használtam ezt a technológiát, mivel felhasználása közben rájöttem a korlátaira. Nem tudom van-e olyan technológia, ami képes a magyar nyelvet olyan rugalmasan kezelni, mint az angolt, de én kutatásaim során sajnos nem találtam. A tapasztalatok is azt mutatják, hogy a magyar nyelvet még nem sikerült olyan jól kezelni, mint az angol nyelvet.
Ezért úgy gondolom, hogy tökéletes kutatási téma lehetne egy olyan csetbotokra specializált nyelvet létrehozni, amely az agglutináló nyelveket is képes rugalmasan kezelni. Ezen kívül, ugyancsak a magyar nyelv típusa miatt előjövő probléma miatt találni egy módszert arra, hogyan találjuk meg a különböző szavak szótöveit. A kulcsszó keresés, az élet több területén is hatalmas segítség lehet a csetboton kívül is.

3. Keresési algoritmus

Az általam kialakított keresési algoritmus nem minden esetben működik jól. Úgy vélem az

alaposabb teszteléséhez külsős tesztelők bevonására is szükség lenne. A tesztelők által felfedezett, és dokumentált problémákat típusonként összegyűjtve esettanulmányok is összeállíthatók lennének, amelyek sokat segíthetnének az algoritmus további tökéletesítésében.

Úgy gondolom, hogy amit szerettem volna, azt sikerült megvalósítanom, de a probléma, amivel szembe kerültem jóval bonyolultabb, mint eleinte gondoltam. A jövőben szeretnék továbbra is foglalkozni ezzel a területtel, mert érdekesnek, szakmai kihívásnak találtam.

VI. Irodalomjegyzék

Balogh, A., Halácsy, P., & Szalai, A. (2014.). Letöltés dátuma: 2019. március 3., forrás: ftp://ftp.mokk.bme.hu/Language/Hungarian/Freq/Web2.2/

Drift, SurveyMonkey, Audience, myclever & Salesforce. (2018). *The 2018 State of Chatbots report.* Letöltés dátuma: 2019. április 8., forrás: https://www.drift.com/wp-content/uploads/2018/01/2018-state-of-chatbots-report.pdf

GeeksforGeeks. (é. n.). Letöltés dátuma: 2019. 05. 1., forrás: https://www.geeksforgeeks.org/print-all-possible-combinations-of-r-elements-in-a-given-array-of-size-n/

Havas, F. (1974). *A Magyar, A Finn, És az Észt nyelv Topológiai összehasonlítása.* Budapest: Akadémiai Kiadó.

Hedley, J. (2018.). Letöltés dátuma: 2019. március 8, forrás: https://jsoup.org/

Horváth, S. (é.n.). *JavaServer Faces 101 - Avagy az első keretrendszer ami szembejön webes Java fejlesztés során.* Letöltés dátuma: 2019. március 1., forrás: Skillversum: https://www.skillversum.com/note/view/13267ab7cb50a6a903fb26faaefafbf47cd623e4

HowToDoInjava.com. (2018. Február 5.). Letöltés dátuma: 2019. május 11., forrás: Convert InputStream to String In Java: https://howtodoinjava.com/java/io/how-to-read-data-from-inputstream-into-string-in-java/

JavaWorld. (2018. December 6.). *What is JSF? Introducing JavaServer Face.* Letöltés dátuma: 2019. április 27., forrás: https://www.javaworld.com/article/3322533/what-is-jsf-introducing-javaserver-faces.html

Kilián, I. (2018). *Objektumközpontú szoftverek építése Projekt és szoftver tervezés.* Pécs: Pécsi Tudományegetem Természettudományi Kar, Szentágothai János Protestáns Szakkollégium.

Magyar Tudományos Akadémia. (2015). *A magyar helyesírás szabályai* (tizenkettedik. kiad.). Budapest: Akadémiai Kiadó Zrt.

Petherbridge, N. (é. n.). Letöltés dátuma: 2019. április 15., forrás: https://www.rivescript.com/

Petherbridge, N., & Overdijk, M. (2016). Letöltés dátuma: 2019. március 27., forrás: https://github.com/aichaos/rivescript-java

Phillips, C. (2018. április 30). *The 3 Types of Chatbots & How to Determine the Right One for Your Needs.* Letöltés dátuma: 2019. május 2., forrás: https://chatbotsmagazine.com/the-3-types-of-chatbots-how-to-determine-the-right-one-for-your-needs-a4df8c69ec4c

Salesforce.com. (é. n.). Letöltés dátuma: 2019. május 2., forrás: What is Heroku: https://www.heroku.com/what

Software Freedom Conservancy. (é. n.). Letöltés dátuma: 2019. május 11., forrás: 1.3 Getting Started - What is Git?: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F

The Apache Software. (é. n.). *What is Maven?* Letöltés dátuma: 2019. április 27., forrás: https://maven.apache.org/what-is-maven.html

The PostgreSQL Global Development Group. (é. n.). *8.3. Character Types.* Letöltés dátuma: 2019. Május 10., forrás: https://www.postgresql.org/docs/9.1/datatype-character.html

The PostgreSQL Global Development Group. (é. n.). *What is PostgreSQL?* Letöltés dátuma: 2019. május 3., forrás: https://www.postgresql.org/about/

w3schools. (2019). *How TO - Chat.* Letöltés dátuma: 2019. április 1., forrás: https://www.w3schools.com/howto/howto_css_chat.asp

w3schools. (2019). *How TO - Popup Chat Window.* Letöltés dátuma: 2019. április 1., forrás: https://www.w3schools.com/howto/howto_js_popup_chat.asp

NYILATKOZAT
az írásmű eredetiségéről

Alulírott **Baráth Péter Gergő (Q0H02C)**, büntetőjogi felelősségem tudatában kijelentem, hogy a **TTKBOT CHATBOT ASSZISZTENS** című írásomban foglaltak saját, önálló munkám eredményei, ennek elkészítéséhez kizárólag a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel, írásomat a Pécsi Tudományegyetem vonatkozó szabályzatainak betartásával készítettem. Tudomásul veszem, hogy a szerzői jogi szabályok betartását a Pécsi Tudományegyetem plágiumkereső rendszeren keresztül ellenőrizheti.

Pécs, 2019 05. 13.

…….

hallgató aláírása