

Pannon Egyetem

Műszaki Informatikai Kar

Villamosmérnöki és Információs Rendszerek Tanszék

Üzemmérnök Informatikus BProf.

## SZAKDOLGOZAT

Fejlesztést segítő háttérrendszer vagy Autocorrect rendszer fejlesztésekhez

Szabó Dániel

Témavezető: Dr. Guzsvinecz Tibor

2021



# PANNON EGYETEM

## MŰSZAKI INFORMATIKAI KAR

### Üzemméternök-informatikus BProf szak

Veszprém, 2021. október 18.

## SZAKDOLGOZAT TÉMAKIÍRÁS

**Szabó Dániel**

Üzemméternök-informatikus BProf szakos hallgató részére

### **Fejlesztést segítő háttérrendszer, avagy Autocorrect rendszer implementálása fejlesztésekhez**

Témavezető: Dr. Guzsvinecz Tibor, adjunktus

#### **A feladat leírása:**

Nem csak alkalmazásfejlesztés közben, de akár internetes kommunikáció során a felhasználó gyakran elgépelhet egy kulcsfontosságú szót. Ennek kiküszöbölésére jött létre az úgynevezett „autocorrect” funkció, amely a felhasználó írásbeli hibáit próbálja kiküszöbölni több-kevesebb sikerrel. Hiába létezik számos „autocorrect” alkalmazás, egyik sem tökéletes, valamint gyakran nem ingyenesek.

A feladat célja egy olyan háttérben futó program létrehozása, ami a ma ismeretes "autocorrect" mobilokon használt gépelést elősegítő funkcióhoz hasonlóan, gyakran használt parancsok felajánlásával segíti elő a fejlesztőt és mindemellett előnyösebb a piacon található hasonló alkalmazásokhoz képest.

#### **Feladatkiírás:**

- Dolgozza fel a témával kapcsolatos eddigi hazai és külföldi irodalmat!
- Határozza meg a szoftverrel szemben támasztott követelményeket!
- Tervezze meg és implementálja a háttérben futó alkalmazást!
- Tesztelje az alkalmazást!
- Készítsen hozzá felhasználói kézikönyvet!

Dr. Vassányi István  
egyetemi docens  
szakfelelős

Dr. Guzsvinecz Tibor  
adjunktus  
témavezető

## Nyilatkozat

Alulírott *Szabó Dániel* hallgató, kijelentem, hogy a dolgozatot a Pannon Egyetem Villamosmérnöki és Információs Rendszerek Tanszékén készítettem az Üzemmmérnök Informatikus végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy a dolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Veszprém, 2021. december 5.



aláírás

Alulírott *Dr. Guzsvinecz Tibor* témavezető kijelentem, hogy a dolgozatot *Szabó Dániel* a Pannon Egyetem Villamosmérnöki és Információs Rendszerek Tanszékén készítette Üzemmmérnök Informatikus BProf végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozat védeésre bocsátását engedélyezem.

Veszprém, 2021. december 5.



aláírás

## **Köszönetnyilvánítás**

Köszönöm Dr. Guzsvinecz Tibor témavezetőmnek azt hogy olyan türelmes volt, készségesen a segítségemre állt és tanácsokat adott.

Hálás vagyok szüleimnek az éveken keresztül gondoskodásért, a támogatásért, és hogy végig segítettek a tanulmányaimban.

Továbbá köszönöm a barátaimnak, hogy a mindennapokban a társaim voltak és támogattak mindvégig.

## TARTALMI ÖSSZEFOGLALÓ

Létrehoztam egy háttérben futó alkalmazást és a szakdolgozatomban ezt mutatom be. Ez a program az érintő képernyős eszközökön már beépített funkciójú Autokorrekt-et imitálja, ami segíti az átlagos ember hétköznapijait. A mai rohanó világban az ember kevesebb energiát fordít a dolgok precizításra és ez megmutatkozik a legtöbbször az elküldött üzenetek helyesírásában a mobil eszközök által. Viszont a programom még otthon, számítógépes környezetben ugyan ilyen segítő kezet nyújt az odafigyeléshez akár hivatalos e-mailezéshez, munkahelyi üzenetváltáshoz, szakmai tevékenységekhez vagy akár csak hétköznapi beszélgetésekhez. Lehetőséget ad, hogy személyre szabhassa a felhasználó a szótár tartalmát. Mindezt úgy teszi, hogy nem zavarja a felhasználót a tevékenységeiben. Kisablakosnak nevezhető form alapú kinézettel rendelkezik ami éppen hogy, nem zavarja a felhasználó látóterét a miközben a monitorra fókuszál. A kusza múltú C# program nyelven íródott Visual Studio segítségével. Miközben magasszintű programozási nyelvet használunk végig, a program alapját alacsony szintű eljárások adják meg. Ezen eljárások használatával háttérben futó alkalmazásként is tud funkcionálni a program szimultán a számítógépes eszközünk használatakor. Mindezen úton végig haladva, a tanulságokat, hogyan tudnánk jelenlegi oktatási rendszerben alkalmazni?

Kulcsszavak: P/Invoke, autocorrect, háttérben futó alkalmazás

## **ABSTRACT**

A background application was created that is detailed in this thesis. The built-in AutoCorrect functionality of touchscreen devices is imitated by this application. With it, the everyday life of users are made easier.. Due to today's fast world, less energy is put into precision and this is often reflected in messages written with mobile devices. However, a helping hand is provided by my application even at users' homes in case of official e-mails, workplace conversations, professional activities, or even just simple, everyday conversations. An option is given to customize the dictionary without disturbing users in their activities. Its appearance is small, form-based and it does not interfere with the user's field of vision while focusing on the monitor. The program was written in C# using Visual Studio. While people usually use a high-level programming language, the application is based on low-level procedures. Using these procedures, the program is able to operate as a background service while the device is used for everyday tasks or for work. In the end, how can the conclusions be incorporated into the current education system?

Keywords: P/Invoke, autocorrect, background application

## Tartalomjegyzék

1	Az Autocorrekt .....	1
1.1	Az okostelefon .....	1
1.2	Mi is az Autocorrekt funkció .....	1
1.3	Előnye - Hátránya .....	2
1.4	Autocorrekt máshol .....	3
2	A vízió .....	4
2.1	Szükség és cél .....	4
2.2	A program tervezete .....	4
2.2.1	Skálázódó fejlesztési módszer .....	4
2.2.2	Tervezés és megvalósítás .....	5
2.3	Kezdeti szint .....	7
2.4	Kezdeti tervek .....	8
3	C# .....	10
3.1	Fejlődése <sup>3</sup> .....	10
3.1.1	Története .....	10
3.2	C# jellemzői és Java összehasonlítása .....	11
3.2.1	Single Inheritance .....	12
3.2.2	Built-in Thread és Synchronization támogatás <sup>7</sup> .....	12
3.3	Visual Studio .....	13
3.4	NET Framework .....	14
3.4.1	Class library .....	14
4	Fejlesztés .....	16
4.1	Karakter regiszter módszer .....	16
4.1.1	Platform Invocation Services <sup>8</sup> .....	17
4.1.2	Hook .....	17

4.2	Windows Form .....	22
4.2.1	Komponensek.....	23
4.2.2	Alfolyamatok.....	28
4.3	Fejlesztési lépések tesztelése .....	29
4.3.1	Karakter regiszter .....	29
4.3.2	Textbox.....	29
4.3.3	Adatbázis .....	29
4.3.4	massWordsUpload .....	30
4.3.5	Redundancia .....	30
5	Program állapota .....	31
5.1	Mostani állapot eredeti tervhez képest .....	31
5.2	Hogyan lehetne tovább fejleszteni? .....	32
5.2.1	Enum .....	32
5.2.2	Multi-button és további klaviatúra blokkok használata .....	32
5.2.3	Adatbázis funkciók.....	33
5.2.4	GitHub .....	33
6	Konkurencia .....	34
6.1	Phrase <sup>14</sup> Expander.....	34
6.2	Összehasonlítás.....	36
7	Felhasználói útmutató .....	38
7.1	Ismerkedés a felülettel .....	38
7.2	Ismerkedés a gombokkal .....	38
8	Felhasznált technológiák alkalmazása az oktatási rendszerben .....	39
8.1	Egyetem első lépései .....	39
8.2	Céltudatos szakmai fejlődés .....	40
	Irodalomjegyzék.....	43
	Mellékletek.....	45



## Ábrajegyzék

1. ábra: Hétköznapi esetekben elforduló félrekorrigálás.....	3
2. ábra: Működés fejlesztési terve .....	5
3. ábra: Kezdeti karakter regisztráló form.....	8
4. ábra: Kezdeti vezérlő form .....	8
5. ábra: Funkciók Fejlesztési terve .....	9
6. ábra: TIOBE Programming Community Index grafikonja a leggyakrabban használt kódolási nyelvekről .....	11
7. ábra: Thread működési felépítése .....	12
8. ábra: DLL importálás .....	19
9. ábra: SetHook megvalósítása.....	20
10. ábra: Regisztrált karakter feldolgozás menete.....	21
11. ábra: Futás közbeni felület.....	23
12. ábra: Programhoz tartozó SQL Adatbázis .....	24
13. ábra: Nyelv kiválasztó SQL lekérdezés .....	25
14. ábra: Hozzáadó SQL lekérdezés .....	26
15. ábra: Felülíró SQL lekérdezés .....	26
16. ábra: Törlő SQL lekérdezés .....	27
17. ábra: New Language form .....	27
18. ábra: Send .....	28
19. ábra: Valós idejű kereső SQL lekérdezés .....	28
20. ábra: Google Autocomplete funkcióval működő keresője.....	31
21. ábra: Jelenlegi szótár kinézet.....	32
22. ábra: Sablon kinézet.....	36
23. ábra: Scratch használat közben .....	40
24. ábra: Jelenlegi szótár kinézet – úgy mint C# form kinézet .....	41

## Táblázatjegyzék

1. Táblázat: C# verziói fejlődése.....	10
--	----

# **1 Az Autocorrekt**

## **1.1 Az okostelefon**

Manapság a lakosság nagy része rendelkezik valamilyen féle okos eszközzel. Bátran ki lehet jelenti, hogy a fiatalkor osztálytól kezdve egészen a nyugdíjas évet régóta élvező idősök háztartásában is előfordulnak. Ilyenek lehetnek az okostévék, okosórák, akár még az okoshűtő is könnyen szóba jöhet, viszont ami jelen esetben meghatározó szerep volt, az okostelefon. Ezek az eszközök számos lehetőséget biztosítanak, hogy fel tudjuk venni a lendületet a mai rohanó világgal. Lehet kalória számolást végezni, lépést számolni, alkoholszintet követni, hogy szabadna-e az adott alkoholmennyiség után volán mögé ülni (nem) és érdekesebbnél érdekesebb, viccesebb alkalmazásokat lehet találni.

Legalapvetőleg és elterjedtebben használt alkalmazás azok a kommunikációs alkalmazások. Facebook által létrehozott Facebook Messenger, Skype, Instagram és még sorolhatnám ezeket a programokat. Bár megtalálhatóak a számítógépeken is más-más köntösben, de az egyik igazi különbség az okostelefon és a személyi számítógépek között az az autocorrekt funkció.

## **1.2 Mi is az Autocorrekt funkció**

Autocorrekt funkció (Autocorrect function) egy valós idejűleg, háttérben működő szöveg kisegítő, kiegészítő funkció ami szöveg létrehozása közben egy belső adatbázis szerint megvizsgálja az addig beírt szót és segít ajánlataival helyettesíteni, esetleg ha még nem fejeztük be a szó begépelését ki tudja egészíteni azt. Igaz, számítógépes szerkesztő programok is rendelkeznek ilyen funkciókkal, viszont ennek a működése valós időben történik a szerkesztő programok esetében, utólagosan jelzi nekünk az esetleges helyesírási problémát.

Ebben mi a jó? Érintőképernyős telefonokon, tableteken használatos, de a legtöbb érintőképernyős eszközön való gépelés, ha szükséges, sem feltétlen tud teljes mértékben kiszolgáltatni egy billentyűzetes periféria adta funkció hatékonyságát, függően a környezeti és egyéb tényezőktől. Ilyenek lehetnek erős szellőkések által instabil helyzet, ami megnehezíti a felület nyomkodását, kijelző koszos, karcos ezáltal nehezebben érzékeli a felületre adott nyomást vagy esetleg illuminált állapotban kalandosabb egy ilyen szintű szövegbevitel.

Az ilyen problémák esetre hozták létre az Autocorrekt funkciót, ami nehéz helyzetekben segít egy-két karakter bevitele után megállapítja milyen szó leírása a szándékunk vagy akár akkor is nagy segítség tud lenni amikor gyorsan szeretnénk leírni a mondanivalóinkat és így hosszabb szavaknál elég a megfelelő mennyiségű karaktert leütni hogy a funkció felajánlja azt amit

szeretnénk. Rengeteg szituációs helyzetben pozitív mégis sok előnye mellett hátránya is van jócskán.

### 1.3 Előnye - Hátránya

Fentebb említettem pár szituációt milyen helyzetekben van nagy előnye. Nagyon sok más módon is előnyhöz tud a felhasználó jutni, ha egy kicsit jobban is ki ismeri ezt a funkciót. Ha telepít a telefonra több elérhető nyelvet gépeléshez akkor azokon a nyelveken is egyszerűen kitudja önmagát segíteni vagy akár éppen folyamatban lévő nyelvtanuláshoz is hasznos a szavak helyessége ellenőrzése érdekében, így a nyelvtudást is lehet csiszolni könnyűszerrel.

Természetes módon, mint minden program, ez is rendelkezik némi szépséghibával. Amíg más programok esetében a kisebb problémákat el tudják fedni, ez esetben fele annyira sem egyszerű. Megashare az a probléma is az elsődlegesen felajánlott szónál, hogy oda nem illő, mégis az ember, már megszokásból alkalmazza és így kevésbé érthető lesz a mondanivaló. Jelen esetemben azt szerettem volna írni, hogy “Megeshet” mégis a program utólagosan felajánlott szava a “Megashare” lett, ami már ironikusan is tökéletesen reprezentálja a hiba létezését. Erre a problémára több kifejezés is létezik.

Az egyik a “Cupertino effect<sup>1</sup>”, ami az adott szó használatakor egy teljesen más jelentésű szót kapunk ajánlatnak és a nem-odafigyelés végett használjuk őket hibásan. Nevét onnan kapta, hogy “co-operation” kötőjeles verzióban létezett a szótárakban mégis egyben írva “Cupertino” szót kapták meg amit sokszor használtak véletlenül. Cupertino egy Californiai város, ezért létezhet a felajánlott szavak között, nem mellesleg az Apple Inc. fő központja található meg itt, ezért sokszor metonímiaként használják a cég irányába a Cupertino szót.

Másik kifejezés a “Scunthorpe problem<sup>2</sup>”, mely abból a szerencsétlen helyzetéből adódik, ahol a kereső motor és a spam filter kiszűri úgy mint, obszcén vagy elfogadhatatlan kifejezés. Ezek a szavak sokszor csak egy egyszerű rövidítés, nevek, vagy csupán egy technikai kifejezés sajnálatos rosszul csengő hangzása miatt történhet. Ennek forrása 1996-ra nyúlik vissza egy kis városba, ahol is az AOL (American Online) Amerikai webportál és online service provider meggátolta, hogy az adott város lakosai email-t készítenek, mert a városnév tartalmaz egy obszcén kifejezést.

Általános hibának tekinthető, amikor a hétköznapiakban, hétköznapi üzenetváltás keretén belül felajánlott szó bár megtévesztően hasonló, mint eredeti szándékunk volt, mégis teljesen más értelmet ad az üzenetnek és így rengeteg humoros üzenetváltás jöhet létre. (1. ábra)



1. ábra: Hétköznapi esetekben elforduló félrekorrigálás

## 1.4 Autocorrekt máshol

Rengeteg IDE (integrált fejlesztői környezet) funkciója között szerepel a command suggestion, aminek hasonló a célja mint az autocorrect. Ez a gyakorlatban úgy néz ki, hogy az adott kifejezést elkezdjük begépelni és egy amolyan command suggestion felajánlja minden hasonló nevű függvényt, változót és egyéb a könyvtárban lévő és egyező kifejezést. Ez nagyban megkönnyíti a fejlesztők munkáját, könnyebben kereshető a függvények nevei és paramétereik azonosíthatóak, hogy a megfelelő verziót használhassa.

## **2 A vízió**

### **2.1 Szükség és cél**

Weboldal készítőként, tanulmányaim folyamán és autodidakta önképzéseim alkalmával is sokszor szembesültem annak igényével, hogy IDE-n kívüli program esetében szükségem lett volna parancs kiegészítés segítségére a hatékonyabb munkavégzéshez.

Abban az esetben ha egy weboldal belső kódját szükséges lenne kiegészíteni esetleg bővíteni, akkor megeshet, a körülményeket tekintve csak egy Notepad++ programunk a leghatékonyabb eszköz ehhez. Egy hirtelen kapott kód szívettség béli átnézéséhez is elég egy efféle program, egy felületes segítségnyújtáshoz úgyszintén, viszont a hosszabb komplexebb kódok esetében nehezebb “fapadosabb” megoldásokat tudunk alkalmazni ilyen környezet esetében.

Ha a helyzetet nem tudjuk jobbá, egyszerűbbé tenni alkalmazkodnunk kell hozzá. Felmérni mire van szükségünk, megalkotni vagy egy már meglévő rendszert tovább fejleszteni, átalakítani úgy ahogy számunkra a legmegfelelőbb legyen.

Ezen túl is, a hétköznapiokban használt szövegszerkesztő és üzenetküldő programok esetén is, ugyan úgy ahogy mobilon, ugyanúgy számítógépen is szükségünk lehet egy nyelvhelyesség ellenőrzésre, vagy akár csak egy szimplán elfeledett kifejezés kiegészítésére. Így hát adott a kérdés, miért ne lehetne mind a két fél számára segítséget nyújtani?

Ezt a két szükséges igényt fuzionálva is létre lehet hozni egy programot speciális változatok nélkül. Nem szükséges semmi féle megkötés. Ez a két igény, több eshetőségre is megoldást tud nyújtani. Írók, nyelvfordítók, programozók fejlesztők, sitebuilder vagy bármilyen szakmabeli akinek szükséges a saját szakkifejezéseit könnyen elérhetőként leírni, például egy doktor a latin kifejezéseket. Ezeket a saját “nyelveket” külön lehet választani és akár aktív használat közben egy kattintással átváltani más nyelvre ha szükséges.

### **2.2 A program tervezete**

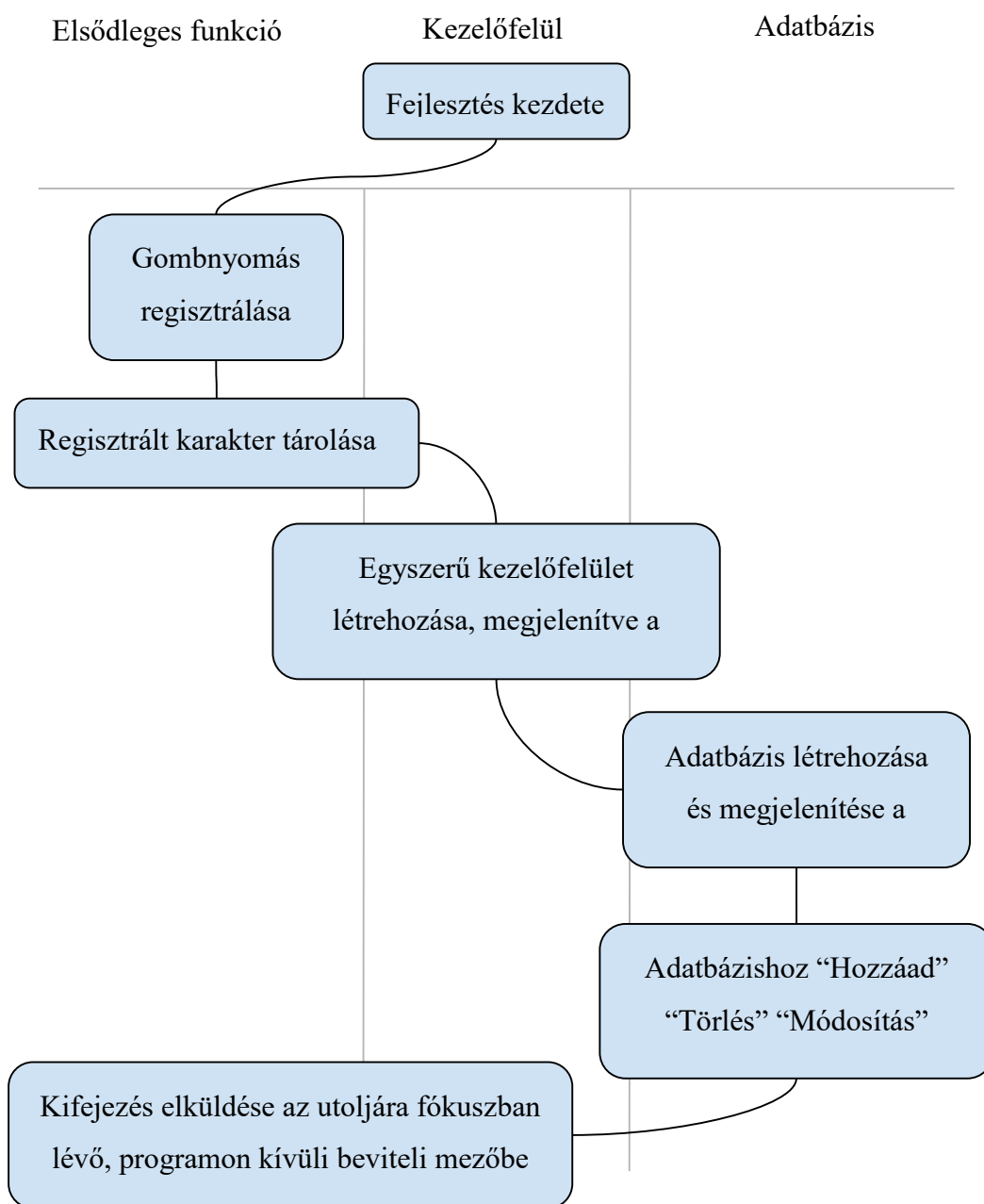
#### **2.2.1 Skálázódó fejlesztési metódus**

Fejlesztés kezdetekor fontos döntés volt lépcsőzetesen felépíteni fejben a program minden részét. Továbbá fontos volt számomra, hogy átgondoltan tervezzem meg a funkciókat, nézeteket, lehetőségeket aszerint hogy minél optimálisabban, minnél kevesebb funkciók módosuljanak így kizárva a nagyobb összeférhetetlenség esetét egy modul implementálásakor. Ehhez arra volt szükség, hogy lépésről lépésre hozzam létre ki a legalapvetőbb szintű

funkcióktól, a többlépcsős tervezetig, aminek a végén egy fluidan működő észrevehetetlen program lesz, ami egy szimpla szó ajánlaton túl, többet is tud nyújtani.

## 2.2.2 Tervezés és megvalósítás

Programozási nyelv használatát tekintve, egy rövid gondolkodási idő után a C# mellett tettem le a voksomat. Magas Szintű nyelv ahhoz, hogy már meglévő és működő megoldásokat tudjak felhasználni a programomhoz a hatékonyság nevében. (2. ábra) 1. ábra



2. ábra: Működés fejlesztési terve

Tervezésnél elsődleges célom az volt, hogy egy olyan metódust tudjak létrehozni, ami által képes legyek regisztrálni a leütött billentyűzeteket Form felületen kívül is. Kutatásaim során többfajta megoldást is kipróbáltam, Keylogger és Hook rendszer.

Keylogger: Teljesebb kifejezése “Keystroke logging”, egy művelet ami regisztrálja / naplózza a leütött gombot. A felhasználók legtöbb esetben nem tud róla, hogy épp megfigyelés alatt áll. Ekkor adatokat gyűjthetnek a felhasználóról. Funkciót felhasználva naplózó programokat lehet létrehozni és a bevitt adatokat fel tudja dolgozni és tárolni.

További veszélye ugyanúgy vonatkozik a Hook rendszerre szintén. Egyszerűen használható viszont annál nyersebb módon lehetett kezelni a bejövő adatokat. Hook rendszer felfedezésekor, annak funkciói és kezelése célirányosabb, és átláthatóbb volt és számomra optimálisabb volt ezt választani.

Mint említettem a Keylogger veszélyeinél, ami ez esetben is igaz a Hookra is igaz az a Keylogging, felhasználási területét tekintve.

### **2.2.2.1 Keylogging**

Ugyan úgy ahogy egy belső céges rendszerként lehet figyelni a dolgozók munkáját keylogginggal, ugyan úgy tudatlanul áldozatai lehetünk otthoni számítógépünkön egy Keylogger kémprogram. De a jelen időnkben a valóságban ténylegesen valós-e a veszély?

Manapság ha esetlegesen hackerek támadó szoftvert használnak sokkal nagyobb kár és mennyiségű adatlopásra alkalmasabb programot tudnak futtatni és akkor inkább a böngésző adatait másolják le ami által az összes bejelentkezési adat könnyűszerrel kimenthető. Ha tényleg csak kis mennyiségű adatlopás lenne a cél akkor a manapság használt “cookie”-kat használnak ilyen célokra.

Ezen veszélyek kivédésére hatékony módszer a Two Factor Authenticator (2FA) kétlépcsős azonosítás, amihez járul telefonos SMS béli belépési megerősítés egyszer használatos kóddal vagy egyéb erre a célra létrehozott program ami közvetlen hozzáférésként érzékeli a belépési szándékot és generálja szintén egyszer használatos kódját. Nyilván ez a módszer is megkerülhető abban az esetben ha az illető 2FA-t használó eszközét valós-időben figyelik meg. Léteznek oldalak ahol többszintű megerősítés is lehetséges.

Jelenleg az egyik leghatékonyabb módszer jelszavak és azonosítók védelmére az a jelszókezelő szoftver. Ezek általában böngésző kiegészítő programként telepíthetők a számítógépünkre, ami regisztrációkor egy mester jelszót kér. Ennek a jelszónak merőben különböznie kell az általában ránk jellemzően használtaktól illetve bonyolultsági szintje is

elvárás, ami azt jelenti, hogy számokat és speciális karaktereket illetve váltakozó kis és nagybetűk egyszerre szerepeljenek benne sok karakter hosszan. Ezen program használata lehetővé teszi, hogy az adott oldalakhoz elmenti a bejelentkezési azonosítót és ha szükséges a program maga letud generálni speciális kritériumok szerinti véletlenszerű kódot amit nekünk nem kell megjegyezni mert a program bejelentkezéskor magam kezeli mindezt.

#### **2.2.2.2 Megjelenítés**

Észrevehetetlen, de háttérben futó folyamatként működne, hogy ne zavarjon bele az aktuális tevékenységekbe. Egyszerű ki-be kapcsolhatóság jellemezné annak érdekében, hogy a nem-szükséges esetekben. Aktív állapotban, mikor feljönnek a szó ajánlatok, a szövegbeviteli mező alatt jelennének meg egymás alatti sorokban. Ennél a felsorolásnál megjelenik egy kisebb icon amivel ki lehet kapcsolni. Továbbá a tálcán egy rejtett iconnal megnyitható a program beállítás része.

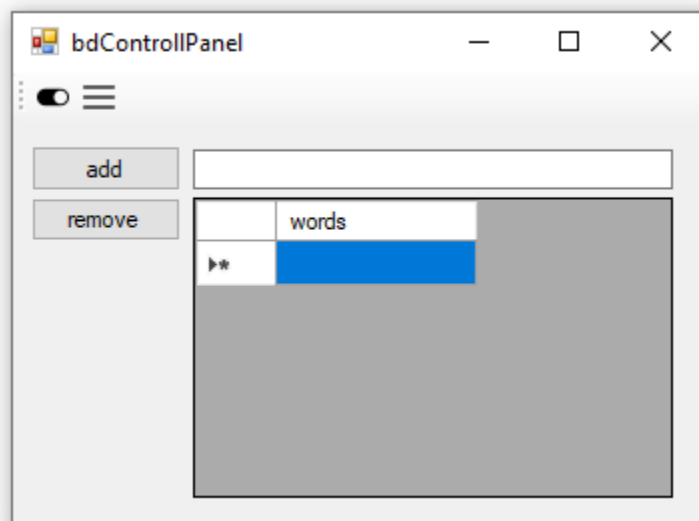
#### **2.2.2.3 Felületek**

Program beállítás, egy teljesen hétköznapi Windows felület jelenik meg menü sávval, egyszerű és hatékony kezelőfelülettel. Lehetőséget kapunk arra, hogy aktív használat nélkül töltsünk fel szavakat az adatbázisunkba illetve módosítsuk, töröljük azokat.

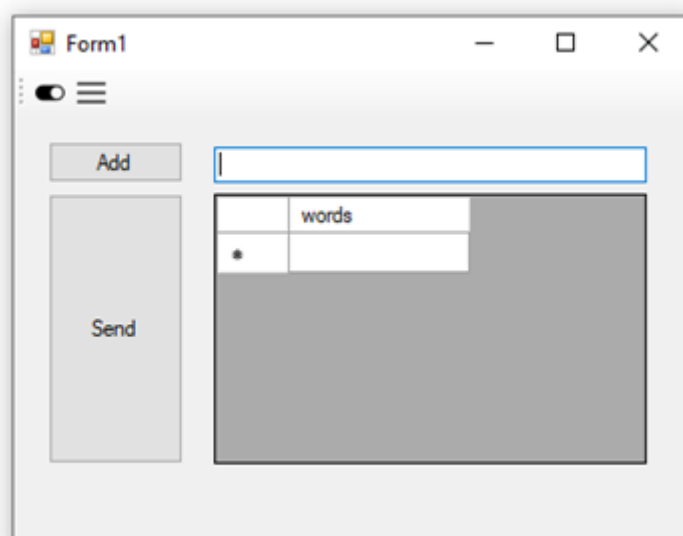
### **2.3 Kezdeti szint**

Alapszintű funkciók megalkotása volt az elsődleges célom. Itt még nem szó szerkezet eltárolása a cél hanem egybefüggő szövegek a Hook által regisztrált és összefűzött valós szövegek kialakítása. Adatbázis szinten csak minimális adatbázis módosítás lehetséges pár gomb használatával a Beállítások felületen. Egyszerű form felület adja a lehetőséget hogy egy gombnyomásra egy, a már adatbázisban lévő elemre kattintva a textbox-ba tudjuk másolni az adott kifejezést, amit "Send" gombnyomással el tudjuk küldeni az utoljára fókuszban lévő program beviteli mezőjébe.(4. ábra) (3. ábra)





4. ábra: Kezdeti vezérlő form

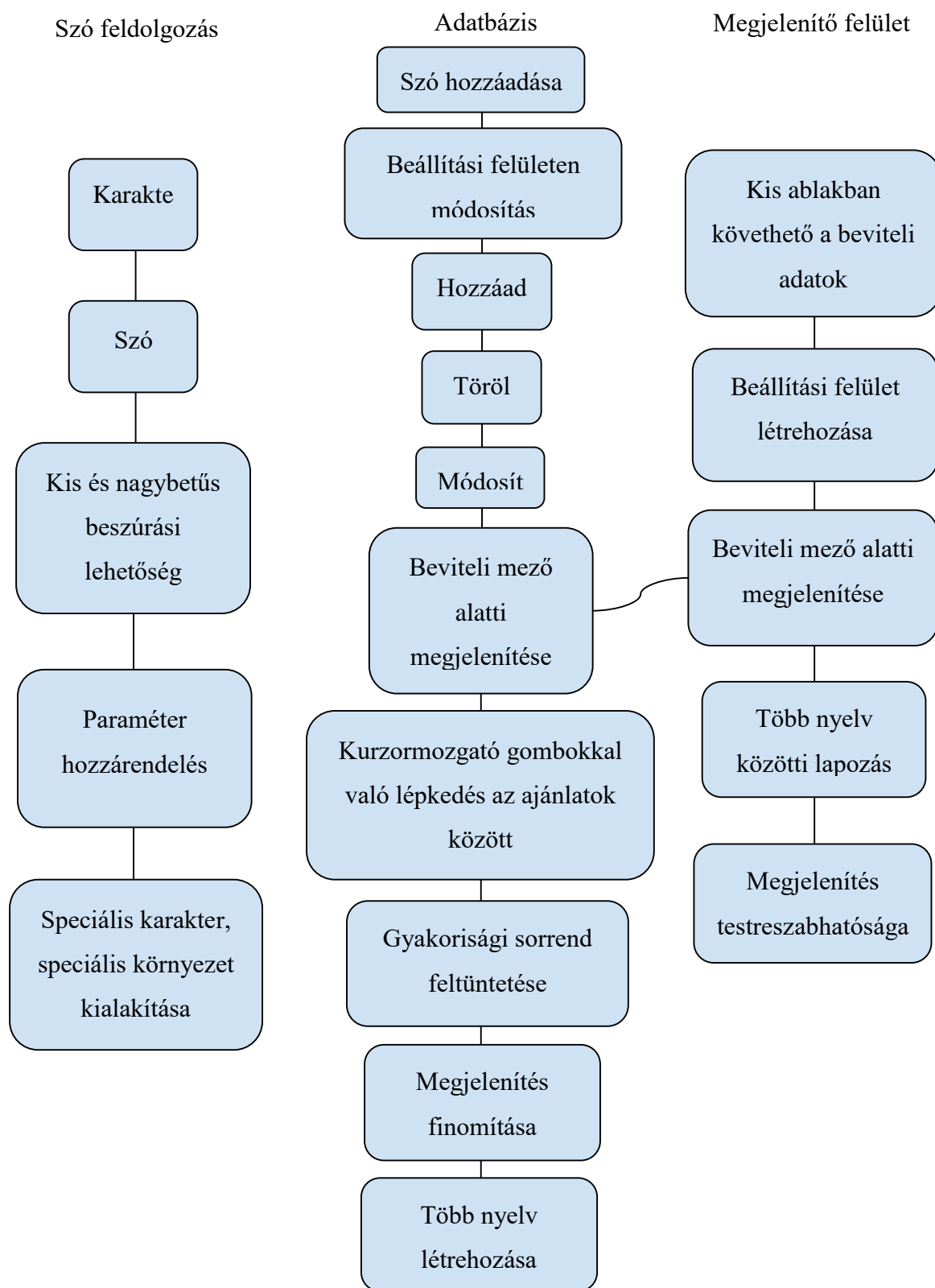


3. ábra: Kezdeti karakter regisztráló form

## 2.4 Kezdeti tervek

Végső rendszerben a kifejezés tárolása szempontjából több lehetőségben adott: Szó, Paraméteres kifejezés programkódokhoz, Speciális karakterekkel ellátott kifejezés például: `<HTML> </HTML>`. Form felületek személyre szabhatók kinézet ügyileg egy külön beállítási funkcióval. Adatbázis feltölthető bizonyos formátummal. Ajánlott szavak közvetlen a bemeneti mező alatt jeleni meg első 5 helyen a leggyakrabban használt kifejezés az adott bemeneti

értékek és abc sorrend alapján. Főmenü ablakban lapozással választható az aktuális nyelv amihez szint is hozzá rendelhetünk megjelenítéshez így véletlen se eshetünk olyan hibába hogy más szótárat használunk mint amit valójában szeretnénk.(5. ábra)



5. ábra: Funkciók Fejlesztési terve

## 3 C#

### 3.1 Fejlődése<sup>3</sup>

1.0	Első hivatalos verzió következő lehetőségeket nyújtotta: attribútumok, operátorok és kifejezések , utasítások, felületek, jellemzők, események, osztályok, struktúrák
2.0	3 évvel később bővítették: iterátok, részleges osztály típusok, anonim függvények, null típus, típus paraméterek, getterek és setterek, statikus osztályok
3.0	2007-ben: anonim típusok, lambda, részleges osztály metódusok , inicializátorok
4.0	beépített referencia típusok, megnevezett és opcionális paraméter, beágyazott együttműködő típusok
5.0	aszinkronos programozható típusok
6.0	statikus direktíva, when catch/case esetben, tulajdonság inicializáló, body-t definiáló kifejezés, null-conditional operátor, nameof kifejezés
7.0+	out változó, tuples és discards, tervezési minták, lokális funkciók, numeric literal syntax improvements, throw kifejezések, aszinkronos main függvény, alapláncolt litáris kifejezések, privát és protected
8.0	csak olvasható tagok, alapláncolt felület metódusok, switch kifejezés, tuple minta, positional minta, using deklaráció, lokális statikus funkciók, null referencia típusok, aszinkronos streams, összetartozó null értékadás, nem felügyeleti konstruktor típusok
9.0	record típus, csak setter inicializáló, felső szintű utasítások, minta azonosítás erősítése

1. Táblázat: C# verziói fejlődése

Az évek folyamán folyamatosan fejlődött a nyelv és a lehetőségek is ezáltal bővültek. Megérkeztek a nagyobb funkciók a nyelvbe és alakult ki a ma ismeretes 9.0. (1. Táblázat)

#### 3.1.1 Története

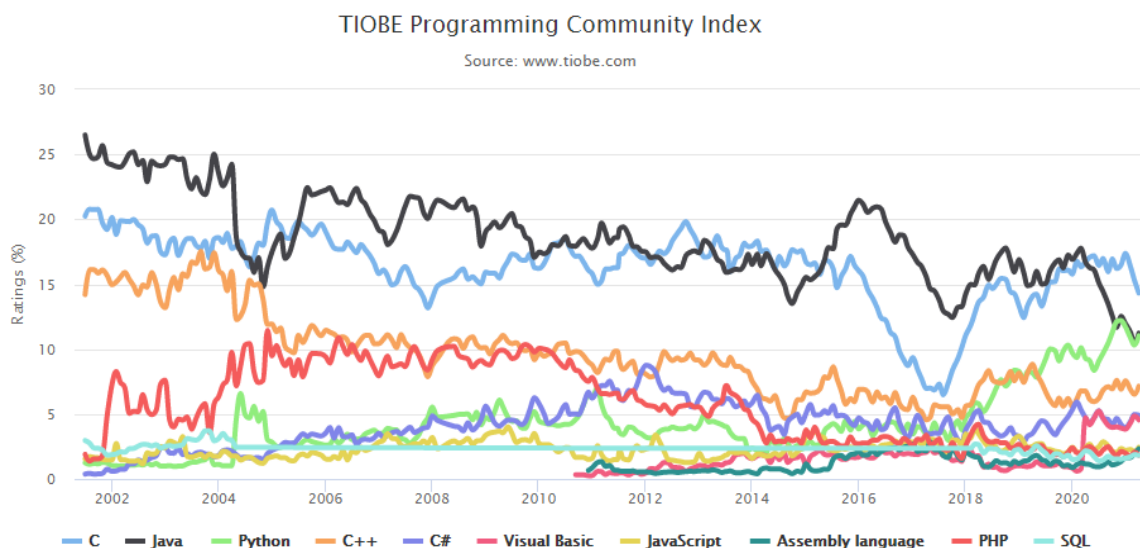
Weboldal 90es években a Microsoft jogi problémákba ütközött a Sun Microsystemmel aki birtokolta a Java licensét. Microsoft próbálta bővíteni saját operációs-rendszerére specifikusan, maga a Java keretrendszert felhasználva. Ez által csak bizonyos rendszereken lehetett volna futtatni és ezzel megsértve a platform függetlenséget. A pert a Sun megnyerte és így a Java-t el kellett távolítani.

Kezdeti fejlesztéskor Simple Managed C (SMC) nyelvet használtak, viszont 1999-ben csatlakozott a csapatvezető pozícióban Anders Hejlsberg - Turbo Pascal tervezője és új programozási nyelv fejlesztésébe kezdtek. Ezt C-like Object Oriented Language (COOL) nak hívták. Cél hasonló volt mint amit a Java esetében történt, egy nyelv aminek a komponenseit objektum orientáltan lehessen fejleszteni és bővíteni. Új névvel, 2000ben lett hivatalosan bemutatva a .NET, ASP.NET és a C#. 4

Publikálás utáni éveiben az Európai informatikai és kommunikációs rendszerek szabványosítási szövetsége (ECMA) standardizálta a C# nyelvet. 5

### 3.2 C# jellemzői és Java összehasonlítása

Mai világban eléggé populárisnak mondható mind a két nyelv (6. ábra), mégis a történelmük és funkcióik hasonlóságaiknak köszönhetően egymás konkurens nyelvének mondhatóak. Nehéz választani mikor, melyik érdemes. Talán szerencsésnek mondhatjuk magunkat abban az esetben, ha belecsöppenünk az egyikbe a tanulmányaink alatt és azáltal sajátítjuk el az alapokat és mélyülünk bele, esetleg tovább haladunk egy ahhoz közeli programnyelv irányába. Bár nem csak ez a két nyelv a terjedt el, idők folyamán. A választék folyamatosan bővült, leginkább magas szintű programozási nyelvek irányába, viszont régóta létező nyelvek a leggyakrabban



6. ábra: TIOBE Programming Community Index grafikonja a leggyakrabban használt kódolási nyelvekről

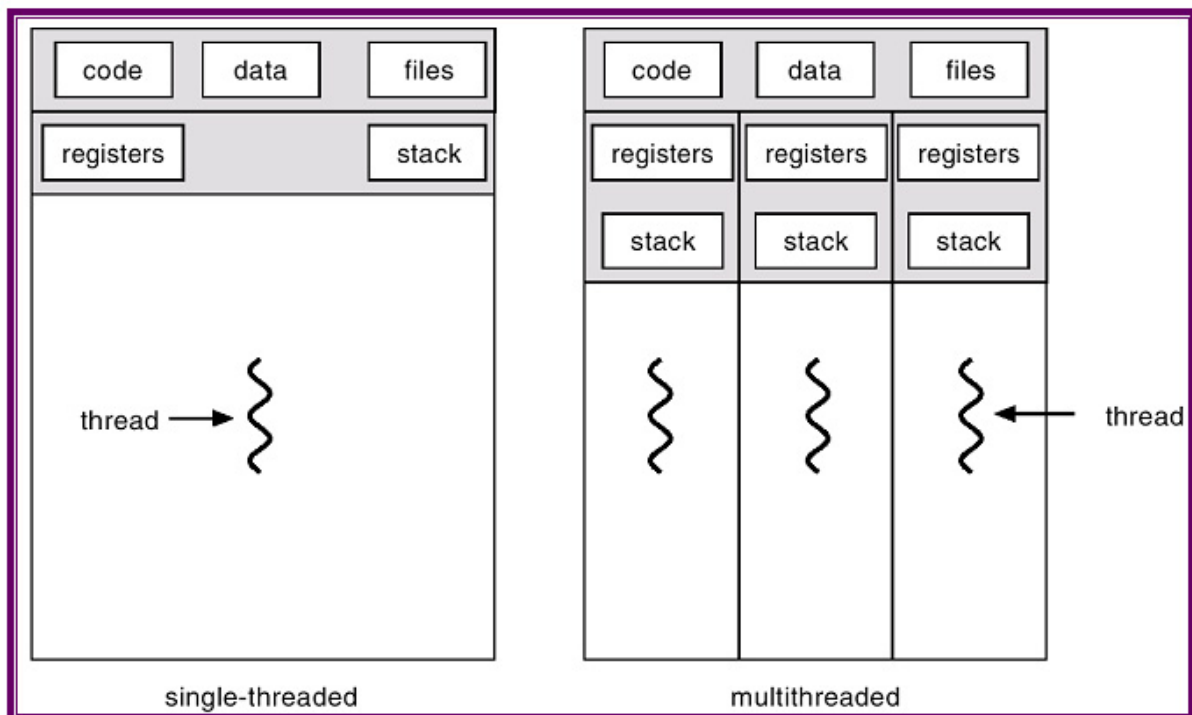
használtak. Ebben a listában szerepel C, Python, C++, Visual Basic régebbi időkben csak Basic, Javascript, PHP, R és SQL.<sup>6</sup>

Mikor C# megszületett a megalkotói ihletet merítették a Javából egyértelműen. A Java fejlesztésekor az Objektum-C tudásához nyúltak úgy szintén ami pedig a C nyelvből lett tovább jelesztve. Elmondhatjuk tehát hogy a C alapja mind a kettőnek. Java előnyeit felhasználták, hátrányait kiaknázták. Így se lett tökéletes a Java, ugyan úgy ahogy a C# sem. Attól, hogy a C# más logika szerint fejlődött tovább még vannak közös elemei amelyek miatt váltak egymás vetélytársává.

### 3.2.1 Single Inheritance

Single Inheritance, egyszeres öröklődés ahol csak egy úton történik öröklődés a parent részéről és megkapja annak funkcióit. Amikor ez az öröklődés végbemegy többszörösen is az nevezzük Multi Inheritance-nek ahol az összes child megkapja a szülő funkcióit. Többszörös a C++-ra jellemző viszont a C# és Javára nem. C++ ban egy virtuális osztály által létrehozhatunk több példányt is. De ehhez ismerni kell minden osztály hierarchiát amit létrehozunk és a jogosultságokkal is tisztában kell lenni különben zavaros lehet a kód.

### 3.2.2 Built-in Thread és Synchronization támogatás<sup>7</sup>



7. ábra: Thread működési felépítése

Mind a 2 nyelv rendelkezik beépített szál kezeléssel, ami egy nagyon előnyösnek mondható. Lehetővé teszi, hogy egyszerre több folyamat fusson egyszerre miközben ebből a felhasználó

nem érez semmit. Amíg egy szórakoztató applikációt használ a mobilján egy ember és a kijelzőt nyomogatja, addig a háttérben “párhuzamos”-nak mondhatóan fut le a többi folyamat is. Ez egyszeri szálon futtatással úgy nézne ki, hogy ha a kijelzőt megérintik, ez után több időnek kell eltelnie, amíg a következő folyamat végig tud menni és azután tud megjeleníteni a változást a képernyőn. Több szál esetében a program futása közbeni metódusok lefutása párhuzamosnak tűnhet, viszont a háttérben a folyamatok időközben más-más időpillanatban fejeződnek be és szintén más időközönként indulnak el. A szálak Process heapjében tárolódnak ezzel megadva a lehetőséget, hogy dinamikusan váltakozzanak a szálak. (7. ábra) Ebben segítenek a Synchronization Objects, Szinkronizációs Objektumok, amelynek 4 típusa van.

- Event - Jelzi a várakozó szálaknak, ha az előtte levő végbement.
- Mutex - Kölcsönös kizárás, egyszerre egy szál férhet hozzá az erőforrásokhoz, többinek várni kell, amíg az erőforrás fel nem szabadul
- Semaphore - Egy meghatározott számú szál mennyiséget szolgál ki erőforrásokkal, így nyomon követhetjük, hányan használják azt.
- Waitable timer - Ezáltal időzíthetjük, mennyi idő után fussanak le az előző folyamathoz képest.

### 3.3 Visual Studio

A Microsoft saját integrált fejlesztői környezete 1997-ben lett kiadva Professional és Enterprise verziókban. Kezdetben Visual F++, Visual C++, Visual Basic és Visual FoxPro nyelvek akkori aktuális verzióit támogatta egészen a 7.0-ig. 2002-ben bejelentették a következő verziót, ami rengeteg újdonságot tartalmazott. Egyik komoly változás az volt, hogy lefelé nem volt kompatibilis, vagyis azon programok, amiket ezt megelőző verzióban írtak, azt nem tudták tovább folytatni. Továbbá ekkor jelentették be a .NET Frameworkot ami által elkezdődött a menedzselt kód fejlesztési módszer. Ekkor a támogatott nyelvek a következők lettek. Visual Basic.NET, Visual C++.NET, Visual C#.NET, Visual J#.NET és megjelent így az ASP.NET is. 2005-ben jelent meg az Express verzió, ami ingyenesen letölthető és használható volt, ami globálisan lehetővé tette, hogy bárki tanulhassa, gyakorolhassa a kódolást, illetve otthoni fejlesztésre. Ugyanebben az évben jelent meg a .NET 2.0-s verziója is. 2010-ben megjelent verziója újabb nagyobb változást hozott. Megújult kezelőfelület, több monitoros lehetőség, programkód nagyítása. Támogatta a Silverlight, Windows 7 platformra való fejlesztést viszont Windows mobile 7-nél régebbi verziójú mobil fejlesztést már nem tette lehetővé. Ekkor a .NET 4.0-s verzióval tartott. 2017-es verzió alatt támogatott lett az Xcode, iOS, watchOS és tvOS és

API-k Xamarin.VS kiegészítőben. További újdonságok: Új moduláris telepítő, részletesebb kivételfelderítő, gyorsabb programindítás és projektkövetés, Live unit-tesztelés, azaz kód írás közbeni tesztelés és eredmény megjelenítés. Run to Click, adott pontig való futtatás és még sok más.

Évek folyamán folyamatosan fejlődtek a nyelvi támogatások, illetve egy idő után a régi, rendszerek felé is visszafelé kompatibilitási támogatás is megjelent. Jelenleg 2019-es verzió a legfrissebb, ami 2019.április 2-án jelent meg.

### **3.4 NET Framework**

A .NET eredetileg fejlesztőeszközök, szoftverek, és hardver eszközök összességét jelentette de mára a .NET összeolvadt a .NET Framework megnevezéssel. Saját szofverként kezdték viszont céljuk volt hogy standardizálják a szoftver stacket. Ingyenes és nyílt forráskódúvá tették viszont a közösség nem volt teljes mértékben megelégedve a feltételekkel főleg nem a szabadalmaztatással kapcsolatos feltételekkel kapcsolatban. Azóta szorosabban tartja a kapcsolatot

A Framework elsősorban Microsoft Windows operációs rendszereken fut. Rendelkezik egy nagyobb osztály könyvtárral, amit Framework Class Library (FCL)-nek hívnak, valamint biztosítja a nyelvi interoperabilitást, azaz más nyelvek használatát számos programozási nyelven keresztül (ami annyit tesz, hogy mindegyik nyelv tud használni más nyelven írt kódot). A keretrendszerhez íródott programok szoftveres környezetben futnak Common Language Runtime(CLR) néven. Ezeket a olyan virtuális gépek amik több téren is segítik a megfelelő lefutást úgy mint biztonság, memóriakezelés és kivételkezelés. Az ilyen kódokat Managed Code-nak nevezzük. Ezekből az elemekből CLR és FCL áll össze a .NET Framework. Az FCL biztosítja a felhasználói felületet, adatokhoz való hozzáférést, adatbázis kapcsolatot, titkosítást, webalkalmazás fejlesztését, numerikus algoritmusokat és a hálózati kommunikációt.

#### **3.4.1 Class library**

A framework CLI egyik része a .NET Framework Class Library (FCL). Ezeket névterekben hívjuk meg. Legtöbbször az Application Programming Interface-ben használják fel névterekre hivatkozva, általában System.\* vagy Microsoft.\*. Könyvtár rendszerét felhasználva hétköznapi funkciók érhetőek el mint a file kezelés adatbázis kezelés XML dokumentum módosítás. Ilyen könyvtár a Base Class Library (BCL) és sok más egyéb könyvtár, van amelyik specializált CLI-re van amelyik pedig Microsoftra.

Bevezették a Common Type System (CTS)-t amivel definiálták az összes lehetséges adatípust és a CLR által támogatott programozási konstrukciókat és azt hogy hogyan léphetnek kapcsolatba a CLI specifikációival. 4.0ás verziótól kezdve a Dynamic Language Runtime (DLR) kibővítette a CLR-t és így lehetségessé vált hogy a dinamikusán beírt nyelveket lehessen implementálni. A CTS és a CLR ebben a környezetben továbbá segíti a fejlesztőt és megakadályozza a rosszul definiált castolásokat, helytelen metódus meghívást és a memória kezeléssel kapcsolatos problémákat is. A rendszer lehetőséget nyújt akár újabb akár régebbi programok funkcióik elérésére, amik a .NET környezeten kívül mennek végbe. A Component Object Model (COM) részeihez a System.Runtime.InteropServices és a System.EnterpriseServices névterek által lehetséges az elérés. A többi funkcióhoz és a Natív alkalmazásokhoz pedig a Platform Invocation Services (P/Invoke) .NET funkcióval lehet hozzáférni.



## 4 Fejlesztés

### 4.1 Karakter regiszter metódus

Elsődleges célom a karakter regiszter rendszer kialakítása volt. Tekintve, hogy erre épül a program szükséges volt, hogy egy stabil átlátható és modern rendszert használjak. Erre ad lehetőséget a .NET. Közel se annyira könnyű egy ilyet megalkotni mint amilyen egyszerűnek tűnik. Windows Form Application esetében beépített a funkció áll lehetőségre, viszont az én esetemben form független kell, hogy működjön a karakter felismerés olyan formában, hogy a felhasználót ne zavarja munka közben. Problémát elméleti szinten kellett lebontani elemeire és aztán tudtam elkezdni érdemlegesen a kutatást. Ennek eredményeképp lett a kiindulópont, maga a gomb lenyomás rendszer szintű vizsgálata. Tekintve, hogy hardver közeli interakcióról beszélünk “lenyomás”-ként fogalmazva, ez arra ad útmutatást, hogy alacsony szintű programozási módszert kell alkalmazni.

Alacsony szintű programozási nyelv egy olyan programozási nyelv ami alig, vagy nem használ absztrakciót a számítógép utasításkészlet-architektúrájából amik hasonlóak a processzor utasításaihoz. Általában ez a gépi kódra vagy Assembly nyelvre jellemző. Megnevezését is néha az alacsony szintű absztrakció által kapja, emellett “hardverközeli” programozási nyelvként is szokták emlegetni. Ilyen módon előállított programok, mivel rendszer architektúrára optimalizáltak, nem igazán számítanak hordozhatónak. Második generációs Assembly nyelv compiler vagy interpreter nélkül futtatja a kódokat közvetlen a processzoron. Nagyon gyorsan futtathatóak nagyon minimális memória felhasználással. Sokkal több idő szükséges ezzel elkészíteni egy programot mint egy magas szintű programozási nyelvvel. Nyelvhasználat nehézsége a rendszer specifikációs ismeretekben rejlik ezért nehezen használhatóak viszont egyszerűek, szemben a magas szintű programozási nyelvekkel, amik kevésbé hatékonyak, sokkal nagyobb memóriára van szükségük, viszont függetleníteni tudja a számítógép architektúráját és a program futását egymástól. A technológia fejlődésével idővel már megjelentek azok az módszerek amik által lehetőség van ezek az alacsony szintű programozási metódusok használata magas szintű programozási nyelvben. Ezt a P/Invoke valósítja meg számomra. Egyik ilyen módszer az “Inline Assembly”, azaz az Assembly gépi kód bele van ágyazva a magas szintű nyelvbe.

#### 4.1.1 Platform Invocation Services<sup>8</sup>

Platform Invocation Services, rövid megnevezése Platform Invoke (P/Invoke). CLI-hez tartozó funkció. Managelt (felügyelt) kóddal natív hozzáférést kaphatunk a .NET keretrendszer funkciói, típusai és metódusaihoz, de az alacsony szintű operációs rendszer és a nem managed külsős könyvtár rendszerhez már nem és az ilyen tevékenységekhez ún.: engedélyekre van szükségünk. Ezekben az esetekben, a P/Invoke technika teszi lehetővé, ami a struktúrákhoz, meghívásokhoz és funkciókhoz enged hozzáférni. A folyamat a következőképp megy végbe, deklarálunk a nem managed funkciót, egy managed funkció által. Hivatkozni fog a könyvtár elérési útjára, meghatározza a függvény paramétereit és a visszatér a managed típusban, amit a runtime (CLR) implicit módon rendez. Ennek előnye, hogy a struktúra módosítások után is működőképes lesz, amíg a megnevezéseik nem változnak. C++/CLI-vel használhatjuk az implicit módot, az applikáció egyidejűleg használja a managed kupacot pointerekkel és a natív memória területeket. Ha egy nem managed adattípus túl komplexnek bizonyul egy implicit átalakításhoz, a keretrendszer a lehetőséget ad, hogy a fejlesztő maga definiálhassa explicit módon ezeket az attribútumokat, így elkerülve az implicit módbeli hibákat. Explicit megoldáshoz pedig szükség van a Dynamic-Linked Libraries (DLLs)-re, amit importálni kell és így elérhetővé válik a natív kód. A funkciót itt is fel kell használni explicit módon alkalmazni ugyanis csakis úgy lehet hozzáférni ahhoz a könyvtárrendszerhez, ami tárolja a számomra szükséges Hooking-ot, mivel az általa hozzáfért fizikai gomb lenyomás utáni információt megkapja azelőtt, hogy még más program megkapná ezen adatokat. Hozzácsatolt metaadatokkal pedig definiálni lehet natív kód alapján hogyan akarjuk rendezni az adatokat. Azáltal hogy alacsony szintű programozási nyelvet használ a programozó, elveszti a keretrendszer adott előnyöket mint pld a típus biztonság, vagy a biztonságos memóriakezelést (Garbage Collection). Így elég könnyen előfordulhat memória szivárgás vagy szegmens hiba.

#### 4.1.2 Hook

Az operációs rendszer folyamatosan kommunikál programokkal és azoknak részeivel funkció hívásokkal eventekkel vagy üzenetekkel. az a kód ami ezt kezeli a Hook. Bizonyos tervezési minták is Hook működési elv alapján mennek végbe. Ezen a szinten való kódolásnál egyaránt fontos és adott minden függvény hogy megfelelő legyen az összhang a működésben. Rengeteg előnye van ennek a módszernek viszont nagyon veszélyes is tud lenni ugyan azon okból kifolyólag. A kommunikáció által módosítani tudja a programok lefutásának a menetét anélkül hogy maga a kódon változtatna. Ilyenkor a rendszer teljesítményén csökkenést vehetünk észre

mivel a hook maga is egy folyamatot visz végbe. Felhasználhatósága széleskörű, kitudja bővíteni az eredeti programot akár új funkciókkal általa. Meglehet változtatni a könyvtár szerkezetekhez akár Wrapper Library során. Ezáltal a könyvtár funkcióit áttöltik egy másik könyvtárba ami kompatibilitást nyújt a felhasználáshoz. Rendszer lekérésekhez tudunk hozzáférni, eger és ami számomra fontos és megoldást nyújt a problémámra, klaviatúra gombjai lenyomásakor rendszer felé küldött kódját is eltudja kapni. Ilyen sok lehetőség után egyfajta informatikai etikai kérdés vetül fel, mivel érezhető, hogy milyen nagy mértékben tud befolyásunk lenni egy már futó programra vagy más több programjaira. Akár a felhasználó is veszélyben lehet:

- megfigyelhető lesz az illető,
- információ gyűjtés,
- kódlopás,
- felhasználónevek
- emailezések
- bankártya adatok

Én esetemben a program átláthatósága és funkciója megjelenítése a felhasználó felé biztosítja a működése etikai tisztaságát.

#### 4.1.2.1 LowLevelKeyboardProc<sup>9</sup>

Program vagy könyvtárból felhasználható visszahívó funkció. Felelős azért, hogy minden egyes alkalommal leütött klaviatúra gombjai a programunk számára tudja biztosítani. Itt importálás módszert kell alkalmazni minden hozzá szükséges funkció estében is.(8. ábra

```
[DllImport("user32.dll")]
1 reference
private static extern IntPtr CallNextHookEx(IntPtr hhk, int nCode,
    IntPtr wParam, IntPtr lParam);

[DllImport("user32.dll")]
1 reference
private static extern IntPtr SetWindowsHookEx(int IdHook,
    LowLevelKeyboardProc lpfn, IntPtr hMod, uint dwThreadId);

[DllImport("user32.dll")]
1 reference
private static extern bool UnhookWindowsHookEx(IntPtr hhk);

[DllImport("kernel32.dll")]
1 reference
private static extern IntPtr GetModuleHandle(IntPtr lpModuleName);
```

8. ábra: DLL importálás

#### 4.1.2.2 SetWindowsHookEx<sup>10</sup>

SetWindowsHookEx függvénnnyel használható kizárólag. Kezeli az összes hook típusú funkció folytonos működését láncra fűzés megoldási módszerével. Folyamatosan tudja figyelni a tevékenységeket, amit installálunk rá. Sikeres lefutáskori visszatérési értéke maga a meghívott hook procedúra, jelen esetben a LowLevelKeyboardProc. SetWindowsHookEx Paraméterei a következők.:

- *idHook* – meghatározzuk, hogy a hook típusok közül melyiket akarjuk használni. Jelen esetben - *WH\_KEYBOARD\_LL* = 13
- *lpfn* - típusal előre definiáljuk, hogy a könyvtárból melyik funkciót akarjuk meghívni – LowLevelKeyboardProc - típus
- *hMod* – Általában az esetek nagy százalékában DLL-be importálnánk ezeket a lekéréseket viszont itt csak a programon belül tárolódnak és ezt segít a GetModuleHandle meghatározni.

- *dwThreadId* – 0-ás értékkel egy asztali alkalmazás esetében az összes szálát figyeli, viszont akár beállítható lehetne hogy melyik szálát fontos nekünk

Érdeemes tisztában lenni a hookal kapcsolatos lehetőségeinkkel, mi mást installálhatunk még. Rendszer információk, hardware egér, dialógus üzenetek, message box, menü, görgetősáv, shell applikációk és jelen esetben a klaviatúra megfigyelési modulokat. Legtöbb esetben választhatunk, hogy még a rendszer be érkező adat feldolgozása után vagy előtt kapjuk meg

```
1 reference
private static IntPtr SetHook(LowLevelKeyboardProc proc)
{
    IntPtr moduleHandle = GetModuleHandle(IntPtr.Zero);
    return SetWindowsHookEx(WH_KEYBOARD_LL, proc, moduleHandle, 0);
}
```

9. ábra: *SetHook* megvalósítása

ezeket az adatokat. Érdeemes itt belegondolni, hogy ténylegesen milyen veszélyes lehet egy rossz kezek közé kerülő efféle tudás. (9. ábra)

#### 4.1.2.3 CallNextHookEx<sup>11</sup>

Hook lánc esetében mindenképp kell, hogy belelegyen állítva egy *CallNextHookEx* visszatérési metódus a hook procedúra meghívása alatt. Legfontosabb eleme egy hook láncban ugyanis ennek beállításával fog a hook folyamat képes lenni arra, hogy egy lenyomás után újra készen álljon az információ fogadásra. Paraméterei:

- *hhk* – ez az érték *IntPtr.Zero* kell hogy legyen
- *nCode* – ezáltal fogja tudni a következő hook elem, hogy milyen eljárási módot használjon.
- *wParam* – Milyen típusú hookként kezelje ezt az eljárási módot.
- *lParam* – Egy másik típusú visszatérési érték aminek a lényege hasonló mint a *wParam*.

Ezáltal válik a lánc teljes értékűvé. *CallNextHookEx* biztosítja azt, hogy a hook folytonosan működjön. Beállítjuk egy személyre szabott függvény által *SetWindowsHookEx*-el visszatérési értéként kezelve, hogyan induljon el a folyamat. Az itt paraméterben megadott egyéni függvény végigfut feldolgozva az előzőekben kapott adatokat majd ennek visszatérési értéke, ami által paraméterként használható a *SetWindowsHookEx*-ben, a *CallNextHookEx* ami folytatja a folyamatot.

Viszont egy öngenerált folyamatot is le kell állítani egy programban és ezt a folyamat láncot szakítja meg az UnhookWindowsHookEx<sup>12</sup> ami paraméterében azt az értéket várja amire a legelején meghívódott a SetHook.

#### 4.1.2.4 HookCallback

Szükségünk lesz egy saját funkció létrehozásra annak érdekében, hogy a megkapott információval érdemlegesen képesek legyünk dolgozni a továbbiakban. HookCallback, ebben a funkcióban lesz visszatérési értéként meghívva CallNextHookEx. A feldolgozási folyamatot primitív módon vezettem végig a követhetőség miatt. Felhasználható paraméterek: *nCode*, *wParam*, *lParam*. *nCode* egy int értéket vesz fel. Ha kevesebb, mint 0 értéket ad, akkor

```
1 reference
private static IntPtr HookCallback(int nCode, IntPtr wParam, IntPtr lParam)
{
    if (nCode >= 0 && wParam == (IntPtr)WM_KEYDOWN)
    {
        int vkCode = Marshal.ReadInt32(lParam);
        if (((Keys)vkCode) == Keys.OemPeriod)...
        else if (((Keys)vkCode) == Keys.Oemcomma || (((Keys)vkCode) == Keys.Decimal))...
        else if (((Keys)vkCode) == Keys.Divide)...
        else if (((Keys)vkCode) == Keys.Multiply)...

        || ((Keys)vkCode) == Keys.F11
        || ((Keys)vkCode) == Keys.F12
    )
    {
        Program.globalString += "";
    }
    else
    {
        if (caseStatus == -1)
            Program.globalString += ((Keys)vkCode).ToString().ToLower();
        else
            Program.globalString += ((Keys)vkCode).ToString();
    }
}

return CallNextHookEx(IntPtr.Zero, nCode, wParam, lParam);
```

10. ábra: Regisztrált karakter feldolgozás menete

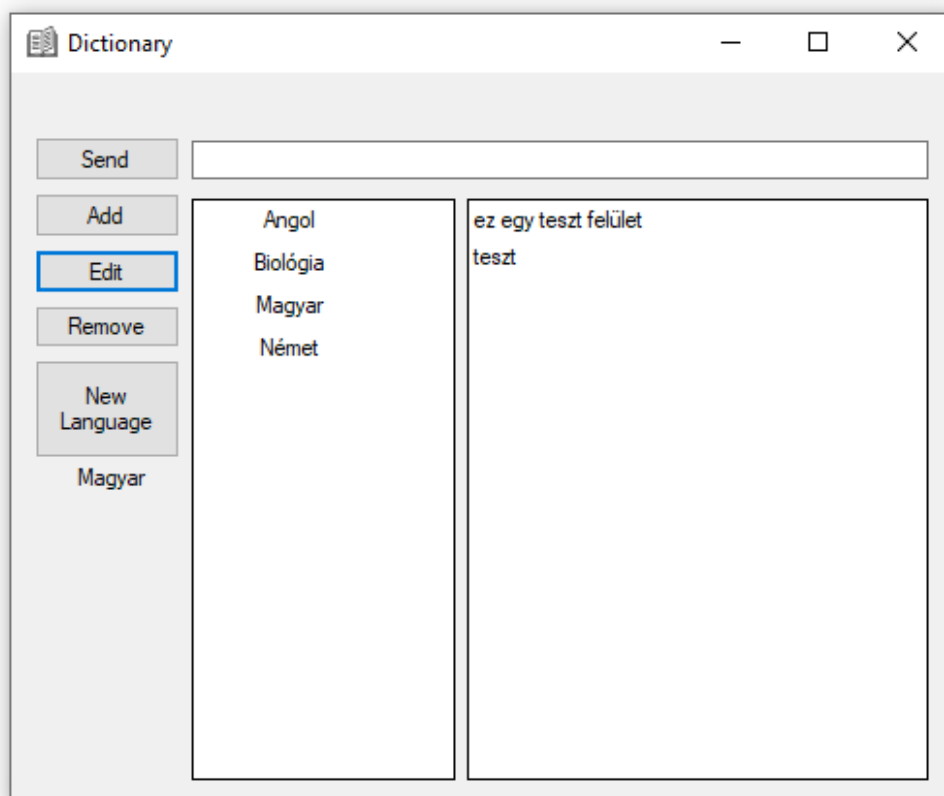
egyenesen a CallNextHookEx-et hívja meg hogy tovább mehessen mivel ilyenkor nem érzékel információt és nincs mit feldolgoznia. *wParam* hordozza a Hook típus béli LowLevelKeyboardProc fajtáját. *WM\_KEYDOWN*, *WM\_KEYUP*, *WM\_SYSKEYDOWN*, vagy *WM\_SYSKEYUP* előfordulása lehet. Én esetemben *WM\_KEYDOWN*-t elegendő volt használni működési kritériumként. *lParam* 32-bit long típusú változó, de mivel unmanagelt folyamatból származik, ezt fel kell dolgozni. Az ilyen adatokat Marshal osztály segítségével dolgozható fel.

Nekünk Marshal.ReadInt32 –re van szükségünk az adat átdolgozásához, hogy aztán int változóként eltudjuk tárolni, *vkCode*-ba. Létezik egyfajta Enumeráció a Keys, amivel betudjuk azonosítani a gombokat. Minden egyes karakter esetében elágazásokkal várjuk és vizsgáljuk meg a kapott értéket majd így a *globalString* értéket itt fogjuk tudni manipulálni. hozzáfűzni az eddig meglévőkhöz karakterláncához. Backspace-val az összefűzött string utolsó elemét lehet törölni. Caps lock is szintén képes működni a mindennapi módján. Minden más jelenleg nem szükséges elem kódja, null értékkel, „” adódik hozzá a többihez.(10. ábra)

## 4.2 Windows Form

Ahhoz hogy több formon is biztonságosan tudjam használni a feldolgozott eddig regisztrált és tárolt kifejezést amellet hogy még mindig aktív a Hook folyamat, Globális Statikus string változóként tárolom el *globalString*. Működés közben több nézet közt is váltogatunk így elvesztve az aktuális értékeket. Ezért használok több változót is Globális Statikusként. A nagy segítség a fejlesztésben Windows Form graphical user interface (GUI), grafikus felhasználói felület. Letisztult, egyszerű felhasználhatósága segítséget jelent a vizuális tervezéshez. Ezt a fajta kényelmi funkciót leginkább az adja hogy a Windowshoz hasonló standard külsővel ruházhatjuk fel, viszont megvan a szabadságunk további testreszabhatóságra. Efféle hatékonyságot Rapid Application Development (RAD), gyors alkalmazásfejlesztésnek nevezzük. Drag/Drop módszer is ilyen technika közé tartozik, ahol elegendő a modult megfogni és elhelyezni a vizuális form alkotó felületen. Az Application Programming Interface (API) a Visual Studio egyik előnyéhez sorolható. Ezt a fajta szabadság részét adja, hogy komponens alapú a rendszer. Több típusú komponens áll rendelkezésünkre így a legtöbb igény pár kattintással megoldható. Ilyen komponens típus az adatfeldolgozó, dialógus, nyomtatásban közbenjáró, menü és eszközkészlet, tároló, általános vezérlők. Eventekkel lehet kezelni a komponens-felhasználó közti interakciókat függvények létrehozásával. Komponens tulajdonságait pedig közvetlen a felületen lehet is állítani, vagy akár futtatás közben is lehet befolyásolni a méreteit, elhelyezését és elég részletesen állítható a kinézet is amivel akár megkerülhető a standard Windows kinézet.

Ez a fő felület ami program futása közben megjelenik. (11. ábra)



11. ábra: Futás közbeni felület

## 4.2.1 Komponensek

### 4.2.1.1 Textbox

Egy textboxot felhasználva jelenítem meg a *globalString* változó értékét. Itt tudjuk nyomon követni a beírt szöveget. Más funkcióknál is segítségünkre lesz, de erre majd későbbekben kitérek.

### 4.2.1.2 Timer

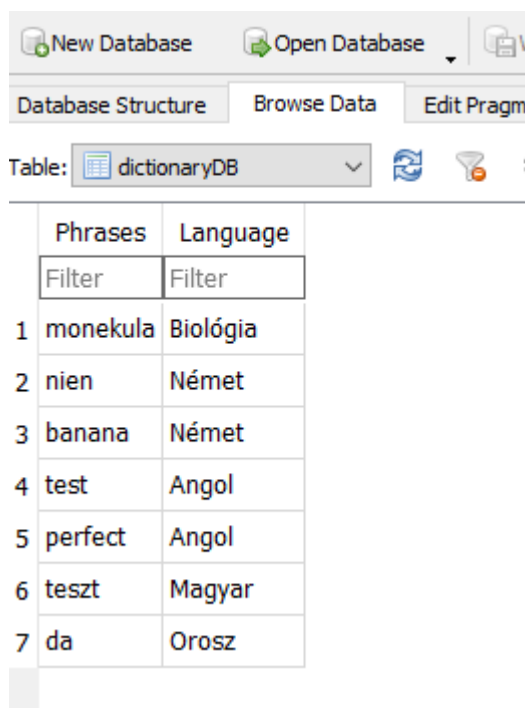
Ún. Timer osztályt felhasználva a program „állandó” módon frissíti a textbox tartalmát. A form betöltésekor bekapcsoljuk a komponenst. Ez idő alatt lehetőségünk van időközönként frissíteni a textboxunkat hogy mindig az aktuális *globalString* jelenjen meg.



### 4.2.1.3 SQLite

Ahhoz hogy minden szót tudjunk megfelelően tárolni szükségünk van egy adatbázisra. Ezt az SQLite szolgáltatja aminek a csomagját a NuGet. SQLite ahogy a nevéből is adódik egy SQL alapú adatbázis ami könnyen implementálható a programjainkba és egyszerű hozzáférést biztosít. Ahhoz, hogy adatbázist tudjunk fejlesztőként létrehozni és beállítani az rendelkezésünkre áll, egy az SQLitehoz készült DB Browser for SQLite program. Egyszerűen használható és kezelhető felületet biztosít az adatbázis kezeléshez és a NuGet package miatt a programfelé tudjuk szolgáltatni. DB.cs osztály kezeli ezeket, létrehoz egy fő kapcsolatot az adatbázissal és függvényeket biztosít az esetleges kapcsolat megnyitás és bezáráshoz a funkciók számára. Adatok szintjén szükség van 2 mezőre. Az egyik, maga az elmentett kifejezés a szótárunkban, másik amivel betudjuk azonosítani melyik nyelvhez tartozik. (12. ábra)

- Adatbázisunk: dictionaryDB
- Nyelvi mező: Language
- Kifejezés tároló mező: Phrases



	Phrases	Language
	Filter	Filter
1	monekula	Biológia
2	nien	Német
3	banana	Német
4	test	Angol
5	perfect	Angol
6	teszt	Magyar
7	da	Orosz

12. ábra: Programhoz tartozó SQL Adatbázis

Az adatbázis kezelő osztály példányosításával megnyílik a kommunikációt a form és az adatbázis között. Ezt kihasználhatjuk a komponenseknél és így tudjuk formálni az adathalmazt.

#### 4.2.1.4 DataGridView

Fontos szerepe van a DataGridView elemnek. Megtudja jeleníteni az adatbázisból lekért adatokat. 2 view elem használatával eltudjuk különíteni a Language rekordjait a Phrases rekordjaitól. Azáltal hogy külön jelenítődik meg a nyelvezet, lehetőségünk van csoportosítani azokat, máskülönben minden egyes Phrases előfordulással párhuzamosan jelennének meg többszörösen. Azzal hogy csak egy rekord jelenik meg a nyelvekből, utat nyit arra, hogy kattintással kiválaszthassuk, melyik nyelvvel akarunk a továbbiakban foglalkozni. Nyelv kiválasztása eltárolódik egy változóba *actualLanguage* és azt felhasználva tovább szűrhető a másik DataGridView tartalma és ott pedig a nyelv alapján lekért rekordokat látjuk.

##### 4.2.1.4.1 datashow

Egy általános adatbázislekérő függvény, amivel bármikor letudjuk kérni az adat, ez komponensek speciális funkcióik esetében egyszerűsíti az adat megjelenítést, esetleges frissítést. Jelen esetben ez a datashow függvény. Egyszerre kezeli a Language mező elmeit csoportosítva és ABC sorban rendezve, és az alapján a Phrases mező rekordjait is. A függvény kezdetben, form betöltésekor, olyan formában hívódik meg ahol a nyelveknél a legelső elem lesz az *actualLanguage*. Ez alapján mondhatni alapméretezett érték az első rekord lesz. Így az első pillanattól kezdve használatra készen áll a program a felhasználó számára. Ez a komponens rendelkezik egy olyan beállítható eventtel ami által, a DataGridView-ban szereplő Phases elemére kattintva kiválasztásra kerül és *globalString* változó megkapja értékét és ezt a

```
//Program.actualLanguage-nek beállítja az első nyelvi előfordulást
Program.actualLanguage = dt.Rows[0][0].ToString();

//Nyelv kiválasztás alapján megjeleníti az adatbázist a dataGridView1-ben
whereLang = "select Phrases from dictionaryDB where Language =" + Program.actualLanguage +
    "'group by Phrases order by Phrases";
cmd = new SQLiteCommand(whereLang, db.GetConnection());
sda = new SQLiteDataAdapter(cmd);
dt = new DataTable();
sda.Fill(dt);
dataGridView1.DataSource = dt;
```

13. ábra: Nyelv kiválasztó SQL lekérdezés

textboxban is le lehet követni.

Alapvető adatbázis kezelő funkciókat el kell látnunk és mivel a felhasználónak közvetlen módon kell interaktálnia azzal, ezért lehetőséget meg kell adni számukra. Ezek a funkciók a

hozzáadás, módosítás törlés és a programunk esetében biztosítanunk kell a nyelvválasztást és új nyelv felvitelét is.(13. *ábra*)

#### 4.2.1.4.2 Add

A textboxban megjelenített szöveg részlet képesek vagyunk egy gombnyomással eltárolni. Ilyenkor az adatbázisba kerül, az aktuálisan használt nyelvként. (14. *ábra*)

```
db.OpenConnection();
SQLiteCommand cmd = new SQLiteCommand("insert into dictionaryDB(Phrases, Language) values('"
    + Program.globalString + "','"+ Program.actualLanguage + "')", db.GetConnection());
cmd.ExecuteNonQuery();
db.CloseConnection();
```

14. *ábra*: Hozzáadó SQL lekérdezés

#### 4.2.1.4.3 Edit

A már elmentett és a DataGridView-ban kiválasztott értékünk, amit megkap ilyenkor a *globalString* változó és egyben meg is jelenik a felső textboxban, Ezután van lehetőségünk az Edit gomb használatára. Kódolás tekintetében úgy oldható meg, hogy egy statikus változót a formok közti ugrálgatás miatt. Ezt egy elágazással megfigyeljük, amint az elágazás megkapja az első állapotú értéket, *modifState*, azt a karaktersorozatot, ami a textbox-ban van, elmenti ideiglenesen egy változóba. majd az elágazásban vizsgált értéket átállítjuk a következő stádiumra. Következő stádiumnál a felhasználónak ezt a karaktersorozatot módosítania kell a textboxon keresztül. Újjonnan lenyomott Edit gombbal tudjuk elmenteni a változtatást. Ez a gyakorlatban kódszinten úgy viselkedik, hogy ekkor az elágazás másikútvonala lesz aktív ahol végbemegy egy UPDATE szintű SQL parancssor és maga az érték felülírás az ideiglenesen eltárolt értékre hivatkozik eredetileg, és az új érték az lesz ami a textfieldben szerepel. (15. *ábra*)

```
db.OpenConnection();
cmd = new SQLiteCommand("UPDATE dictionaryDB SET Phrases='"
    + textBox1.Text + "' WHERE Language ='" + Program.actualLanguage +
    "' AND Phrases='" + modifWord + "'", db.GetConnection());
cmd.ExecuteNonQuery();
db.CloseConnection();
```

15. *ábra*: Felülíró SQL lekérdezés

#### 4.2.1.4.4 Törlés

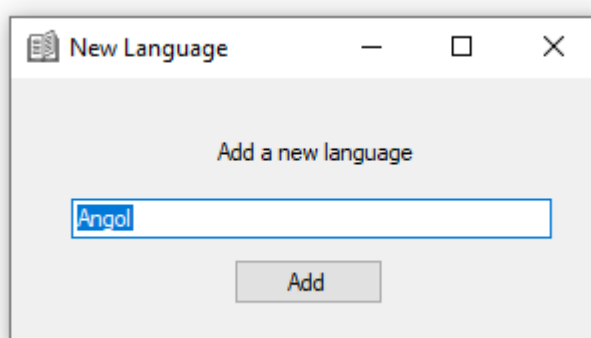
Ugyan azzal a menettel, ahogy az adatbázisból kiválasztunk egy Phrases értéket és bekerül a textboxba az az érték törlésre kerül azon az adott nyelven, ami éppen használatban van. (16. ábra)

```
db.OpenConnection();
SQLiteCommand cmd = new SQLiteCommand("delete from dictionaryDB where Phrases='"
    + textBox1.Text + "'", db.GetConnection());
cmd.ExecuteNonQuery();
db.CloseConnection();
```

16. ábra: Törlő SQL lekérdezés

#### 4.2.1.4.5 New Language

Mivel megeshet, hogy új nyelvet akarunk felvinni, erre is lehetőséget szolgáltat a program. Erre való a New Language gomb. Hatására megjelenik egy új form. Egy textbox lesz elérhető, aminek az alapmértékett értéke mindig az aktuálisan használt nyelv. Amint ezt megváltoztatjuk és rányomunk az Add gombra, visszakerülünk az eredeti Dictionary nézethez ahol az aktuális nyelvindikátor layer-nél látható az előbb felvitt szöveg. Ez a kifejezés még nem kerül bele az adatbázisunkba rögtön, szükség van egy kifejezésre, amit hozzáadunk az adatbázishoz az Add gombbal.(17. ábra)



17. ábra: New Language form

#### 4.2.1.4.6 Send

A legfontosabb funkciója a programnak a Send. Send gomb használatával tudjuk elküldeni az üzenetet a közvetlen előtte bármelyik más felületen aktivált text beviteli mezőre. WindowsForm belüli SendKeys osztály, egyik metódusára hivatkozunk, Send. Ez a paranccsal egy lenyomott gombot tudunk imitálni aszerint, hogy milyen kódot adunk meg neki. Az én esetemben magát

a *globalString*-et adom meg neki ezáltal elérve el hogy akár más programba, vagy weboldalra is lehessen beszúrni szöveget. Ahhoz, hogy elérjem az előző bevitelimezőt csupán csak ideiglenesen be kell zárunk az ablakunkat annyi időre amíg a szöveget átküldjük majd újra meg kell nyitni, hogy tovább tudjuk folytatni tevékenységeinket, figyelve továbbra is a szótárunkat. (18. ábra)

```
this.Hide();  
SendKeys.Send(Program.globalString);  
this.Show();
```

18. ábra: *Send*

#### 4.2.1.4.7 Valós idejű keresés

Másodlagos prioritású funkció a programban a valós idejű keresés. Azáltal, hogy a program kiszolgál minket szótár jelleggel, egy modern megvalósítástól elvárnánk, hogy az adott kifejezéshez könnyen hozzáférjünk rövid idő alatt. A program valós idejű kereséssel rendelkezik. Tekintve hogy összefüggésben van a gomblenyomás – karakterfeldolgozás – formon belüli megjelenítés, szükség is van erre mivel akár egy gyorsíróval is fel kell, hogy tudja venni a versenyt. Egyszerű és kézhez álló megoldást használtam. Mivel a formon belüli megjelenítés esetében a textbox értéke folyamatosan frissül a timer komponens által ezért, felhasználhatjuk azt a fajta tulajdonságát a textbox-nak, hogy megfigyelhető, mikor változik meg a tartalma. *textBox\_TextChanged* eventtel triggerelődik minden egyes ilyen alkalomkor. Maga a keresés folyamata egy egyszerű SQL lekérdezés ahol a *globalString* értéket keressük a Phrases közt mindet a Language leszűrésével. Egyúttal a fenti lekérdezést kell megjeleníteni ugyan abban az időben, hogy a bevitt adat alapján releváns ajánlatokat kapjunk. (19. ábra)

```
cmd = new SQLiteCommand("select Phrases from dictionaryDB where (Phrases like '%" +  
    + Program.globalString + "%') and ( Language like '" + Program.actualLanguage +  
    "')group by Phrases order by Phrases ", db.GetConnection());  
cmd.ExecuteNonQuery();  
sda = new SQLiteDataAdapter(cmd);  
dt = new DataTable();  
sda.Fill(dt);  
dataGridView1.DataSource = dt;
```

19. ábra: *Valós idejű kereső SQL lekérdezés*

#### 4.2.2 Alfolyamatok

Több kisebb alfolyamat zajlik a háttérben a fő funkciók mellett és alatt. *TopMost* attribútummal állítható be hogy minden esetben legfelül legyen a szótárunk és ne vesszen el a szemünk elől,

ha máshova kattintunk. Ablakunk pozicionálást kezdeti betöltéskor beállítjuk azáltal, hogy lekérjük a kijelzőnk szélességét és hosszúságát pixelben majd egy margó méret igazítással egy koordináta objektumot létrehozunk ezáltal állítva új értéket a Location -nek. Egy karaktere szűrő elágazásokat használ az adatbázis hozzáfűzéseknél, elkerülve az esetleges szóköz és félrenyomott 1 karakter, fölösleges hozzáadását. Elvárt hogy a fejléc funkciói is működjenek viszont itt az egyik gomb fontosnak számít ebben az esetben.

Minimalizáló gomb ezesetben a tálca értesítési felületére küldi a programot készenléti állapotba, abban az esetben ha nem lenne már többé szükségünk és egy dupla kattintás után visszkapjuk normális formátumban.

### **4.3 Fejlesztési lépések tesztelése**

Minden funkciónál elsődlegesen primitív módszereket választottam a teszteléshez tekintve, hogy a program alapja egy elég alacsonyszintű programozási módszert használ. Szükséges, hogy teljes mértékben körültekintően járjunk el.

#### **4.3.1 Karakter regiszter**

Minden esetben külön-külön kezeltem le a karakterek kódjait, megfigyelve viselkedésüket és feldolgozhatóságukat. Átlátható voltak a kódok és könnyen tudtam akár beállítani funkciója szerint például, Caps Lock.

#### **4.3.2 Textbox**

Form szinten bárszemmél követhető volt ahogy a textboxban megmutatkozik a bevitt és összefűzött karaktersorozat, mégis külön label-t használtam az éppen aktuálisan tesztelt metódusokra milyen adatokat adnak át egymásnak, illetve párhuzamosan működött egy másik label ahol pedig a kódban való mozgást követtem le X.X:X formátumban.

#### **4.3.3 Adatbázis**

Adatbázishoz köthető módosítások teszteléséhez eleinte egy DataGridView-t használtam és azon belül minden adatot megjelenítve. Ekkor redundás módon jelentek meg a nyelvek mivel minden kifejezéshez tartozik egy. Későbbiekben pedig az adatok szeparált megjelenítése okán is komolyabb szinten tudjuk megvizsgálni az adatok viselkedését a megfelelő működés végett. Nyelv lekövetésére hoztam létre a gombok alatti közvetlen label-t aminek a szövege mindig az aktuális nyelvet írja ki.

#### **4.3.4 massWordsUpload**

Teljesértékű teszteléshez szükségem volt arra a lehetőségre, hogy feltöltődjön az adatbázist szavakkal. NewLanguage-hez tartozó gomb funkciójához commentek közé rejtve helyeztem el massWordsUpload függvénymeghívást. Be kell állítanunk egy útvonalat ahol, elérjük a file-t amiben a szavak vannak tárolva, illetve azt hogy milyen nyelv kifejezéseit akarjuk feltölteni. Feldolgozás soronként történik. Kódban végrehajtódik egy Add hoz hasonló SQL adatfeldolgozás amíg olvassa a file-t. Feltöltöttem English és Magyar nyelvet. A függvény továbbra is elérhető, kódban commentek közt megtalálható és felhasználható.

#### **4.3.5 Redundancia**

Tesztelés szempontjából véleményem szerint a redundáns adat nem hat negatívan a fejlesztés menetére. Könnyebben ellenőrizhetőek az adatot és az esetleges modul fejlesztés befejezésekor, elég akkor rendezni az adathalmazokat.

## 5 Program állapota

### 5.1 Mostani állapot eredeti tervhez képest

A program jelenleg kielégíti az autocorrect funkció lényegi részét. Beírásközben detektálja az inputot és az alapján egy adatbázisból javaslatot tesz a kezelőjének. Megvan a lehetőség, hogy saját szavainkkal bővítsük ezt a bizonyos szó és kifejezés tárat. Hatalmas előnye, hogy nincs ténylegesen megkötve, milyen nyelven vigyünk fel így ezt is személyre lehet szabni. Felvitt adatokat lehet módosítani, eltávolítani. Adatbázisból bármikor átválthatunk bármelyik más addig regisztrált nyelvre. Valós idejű keresésnek köszönhetően mindig megkapjuk a karakterisztikailag hasonló kifejezéseket, amit eltudunk küldeni a legutoljára használt szöveges beviteli mezőbe, legyen az weboldal, jegyzetomb, program vagy bármi más.

Eredetileg az lett volna a célom, hogy maga a használandó felületek beviteli mezőénél jelenjen meg a találati lista, viszont sajnálatos módon jelenleg nem találtam megfelelő módot a megvalósításra. Amíg ez meg nem valósul, addig nem feltétlen van szükségünk gombokkal való navigáláshoz a szótárunkban, illetve a gyorsabb felhasználati módra. Helyett inkább lassabb, de megfontoltabban választhatjuk ki mit szeretnénk. Ennek eredményeképp alakítottam ki azt a fajta kezelőfelületet, ami egy szótár hatását kelti. (20. ábra)



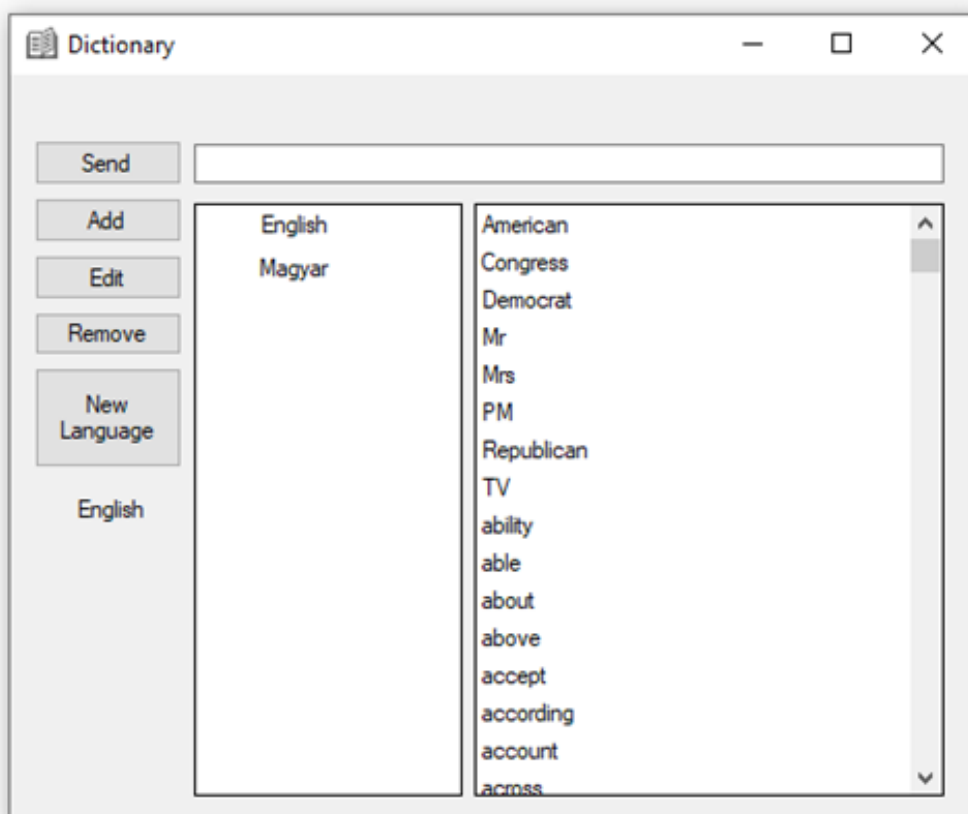
20. ábra: Google Autocomplete funkcióval működő keresője



## 5.2 Hogyan lehetne tovább fejleszteni?

### 5.2.1 Enum

Tekintve, hogy egy komplex szintű megoldást kellett használnom a karakter feldolgozásnál, az adattípusok lekövethetősége miatt többszörös interpretálást kellett végrehajtanom. Ehelyett hosszútávon lehetne az egyszerűbb kezelhetőség és kevesebb programkód miatt bevezetni az enumeráció bővítését. Bár adott a Keys<sup>13</sup> Enumeráció de megvan rá a lehetőség, hogy saját osztállyal string alapú karakterpárt rendeljünk hozzá.(21. ábra)



21. ábra: Jelenlegi szótár kinézet

### 5.2.2 Multi-button és további klaviatúra blokkok használata

Együttal tovább lehetne fejleszteni a hook metódust. Lehetőséget nyitni arra, hogy multi button-t is tudjon érzékelni. Így nem csupán a shift használata mellett lehetne egyszerűen nagybetűket formálni, hanem az egyéb speciális karaktereket is megtudjuk jeleníteni és speciális billentyűk is használható lenne. Mindezen túl akár a vezérlő billentyűket (home, end, pageup, page down) is be lehetne vezetni a megjelenített adatbázis *Phase* mező értékei közt nagyobb léptékű ugráshoz, de akár a navigáló billentyűk használatát is lehetne erre használni egyes léptékekhez.

Nyíl billentyű könnyebbé tenné az adatbázisba való belépést a manuális kijelöléshez is dupla lefele nyíl leütésével. Így az enter gomb használatával pedig ellehetne küldeni a kiválasztott szöveget. Funkció gombokkal pedig értelem szerűen a program funkcióit lehetne könnyen meghívni anélkül, hogy bármilyen gombra is kelljen kattintanunk.

### **5.2.3 Adatbázis funkciók**

Az adatbázis szerkezete megfelelő, viszont azt feldolgozó funkciói menetét lehetne jobban optimalizálni. A folyamatok teljes értékű futtatása, amik esetében aktuálisan megnyitja az adatbázist és a lekérdezés lefutása után pedig a kapcsolatot lezárja. Kézenfekvőbb működéshez elég lenne a program legelején megnyitni a kapcsolatot majd program bezárásakor lezárni azt. Adatbázishoz szükséges lekérdezéseket is tovább szeparálni így csak a szükséges modul futna le egyszer és frissülne az adathalmaz szükségszerűen. Módosító folyamatoknál lelehetne redukálni pársorra az SQL sorainak feldolgozását és végrehajtását és egyszerű paraméterezéssel kevesebb idő lenne a lefutás, tényleges használat alatt.

### **5.2.4 GitHub**

Fejlesztéshez célszerű lenne bevezetni a GitHub használatát bevezetni. Verziókövetést teszi lehetővé, azaz minden egyes változtatást számon lehet tartani, illetve lehetőségünk van a verzióknak elágazni a fő fejlesztési iránytól az esetleges tesztelés és ahhoz köthető változtatásoktól így ha valamelyik része nem működik, vissza lehet térni az eredeti irányhoz.

## 6 Konkurencia

### 6.1 Phrase<sup>14</sup>Expander

Programom főbb funkciójához hasonlítva a legközelebb a PhraseExpander Program áll leginkább. Ez a program ugyan úgy rendelkezik szótár és kifejezés készlettel amit lehet bővíteni és hasonló módon a számítógép használata közben működik. Weboldaluk megnyitásakor egy dolog válik nyilvánvalóvá. Ez a program egy sokkal több és nagyobb kapacitással készült, mint az én projectem.

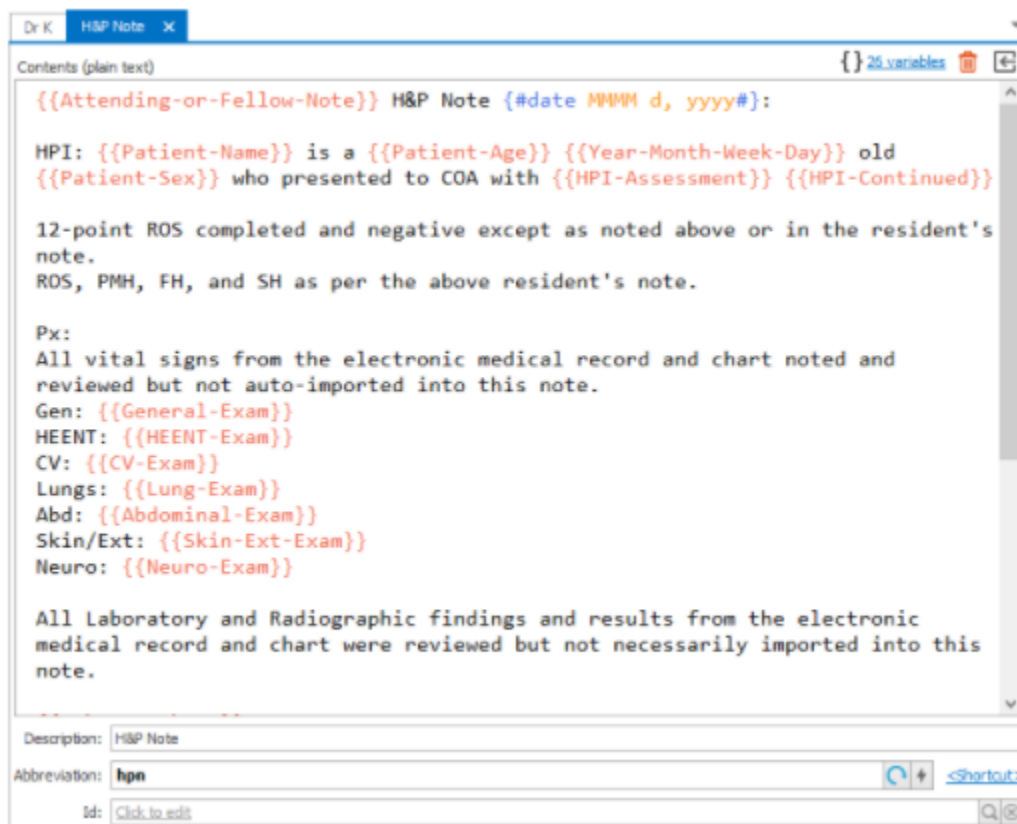
Weboldaluk részletes és informatív. Több csomagban van lehetőség a program megvásárlására. Standard, Professional, Medical. Mélyebbre menve az oldalban a bemutatják a funkciókat és sajátosságait.

- Text expansion – Szöveg kibővítés, akár pár karakter használata után képes feladni javaslatot, ami szóban hasonlít vagy akár a külön hozzárendelt hosszabb szövegű értelmezését is felajánlhat. Itt a kifejezés hozzárendelés több attribútummal rendelkezik. Legutóbbi felhasználási idő, azonosító, leírása, rövidítés, ami lehetővé teszi a gyorsabb gépelést, hipy -> High-Risk Pregnany. Legutóbbi lehetőség által akár hosszabb szövegeket is párkarakter lenyomása után lehet használni, repl24->You can expect a reply within 24 hours. Egy ilyen mondat megidézés 87%-kal gyorsítja meg a begépelés folyamatát. Ezeket a feature-k kihasználását statisztikákon keresztül lehet végig követni. Pár karakter után, ha nem kapjuk meg azt az ajánlást, amit vártunk, gombkombináció lenyomással előidézhetjük az azonos karakterisztikával rendelkező szöveget. Ebbe az adatbázisba beimportálhatjuk a saját szövegeinket is. Abban az esetben, ha helytelenül sikerülne, a beépített autó korrekció kijavítva fogja hozzácsatolni. Lehetőségünk van dokumentumokból, bármilyen szöveges file-ból vagy akár Excelből ki importálni az adatok illetve oda feltölteni azokat, amelyek folyamatosan frissülhetnek.
- SmartComplete - Komolyabb előrelépés e tekintetben az a funkció ahol pár karakter után nem csupán egy szót, kifejezést vagy egy mondatot kaphatunk meg, de előre megformázott hosszabb, többsoros szövegeket is akár. Elkerülhet a gyorsírás közbeni, nagyobb esélyű félregépelések.
- Az autocorrekció ebben az esetben is hasonlóképpen működik, annyi különbséggel hogy a felajánlásokat magunk finomíthatjuk, bizonyos hibák esetén melyek lehetnek

azok a szavak, amiket eredetileg szándékoztunk bevinni, illetve a kis és nagybetűs félregépeléseket is a szöveggörnyezethez viszonyítva is rendezi.

- Másik komolyabb funkciócsoport a sablonok. Érezhetően szempont volt nekik, hogy a felhasználók más szakmában jártasak, főleg Egészségügyi szektorban dolgozók is lehetnek, tekintve az Egészség ügyi csomagjukat. Így lehetőségük van mindenkinek egyaránt saját sablon kialakításra. Rendelkezhet azonos szöveggörnyezettel viszont típus tekintetben pedig dinamikusan változó részeket is tartalmazhat. Használhatóak benne változók is amiket előre definiált logika alapján választatunk ki. Egy fajta felelt választó, rádió gombos opciók, legördülő menüsáv, egyszeri vagy akár többszörösen összetett válasz kiválasztása, egyszerű szövegbeviteli mező vagy akár dátumozás is használható ezekhez a változók kitöltésére. Ezzel egy diagnózis, egy rendelés vagy akár egy hotel szoba lefoglalása is pillanatok alatt megfogalmazódhat szakmai pontossággal.
- Bármilyen külső alkalmazás esetében is úgyszintén lehet alkalmazni, Microsoft Word és egyéb szakma specifikus alkalmazáshoz.
- Nem csupán szövegbeillesztéshez lehet alkalmazni, de még alkalmazásokat is lehet kezelni általa akár, vagy szöveg elemzéseket lehet végezni egy előre meghatározott macro alapján. Abban az esetben ha matematikai képletet jelölünk ki és formai követelményt beazonosítja a program akkor megjeleníti az eredményt.
- Hatalmas lehetőség, hogy megoszthatóak az adataink. Dropbox, One Drive vagy Google Drive felhő szolgáltatás által. Külső program által, de ezen adatok biztonsága is megfelelően kezelve van. Ha szükséges, van lehetőség adatvisszaállításra és jelenlegi helyzet mentésre is.
- Személyes kedvencem, az a vágólap történet megtekintése. Így akár több szöveget is a vágólapon tárolhatunk későbbi időszakra, ha szükséges így nem kell többször megkeresni a forrást.

Az oldalon ingyenesen hozzáférhetőek a sablonok, amit alkalmanként szakemberek hoznak létre és osztják meg mindenkivel segítve a kollegájuk munkáját ezzel. Lehet továbbá találni más szakmák számára sablonokat, az ő munkájukat is megsegítve. Ilyen kiemelt szakmák ügyvédek, informatikai ügyfélszolgálat, vevői ügyfélszolgálat, tanárok és pénzügyben dolgozók. (22. ábra)



22. ábra: Sablon kinézet

## 6.2 Összehasonlítás

Az én termékem ingyenes hétköznapi emberek számára hétköznapi beszélgetésekhez hétköznapi szintű levelezéshez, amihez esetleg szükséges szakmai specifikus és nyelv specifikus kifejezések segítsége, már meglévő szótárakat akár a felhasználó maga bővítheti tovább a lehetőségeit.

Ezzel szemben a konkurens program, bár sokkalta több lehetőséget ad és komolyabb szándékú felhasználók számára, ami által érthető, hogy itt nem ingyenes alkalmazásról van szó. Ez a program egy általános szintű használatra 60 dollár első alkalmi vétel esetében, ami csak az egyszerűbb szöveg kisegítéseket és egyéb kisebb funkciókat tartalmazza, egyszerre telepíthető 3 számítógépre ha Home & Office pack-ként vásárolják meg. Professional amit

már szakemberek számára állítottak össze 150 dollár, ami már tartalmazza az internetes megosztást és a dinamikusabb funkciók használatát. Egészségügyi dolgozók számára már 6 eszközre telepíthető és tartalmaz szakszótárt is, viszont ez már 250 dollár. Első év után a megújítás, már ha azonos verziót akarunk megújítani fele annyiba kerül illetve lehetséges egy nagyobb csomagra váltani.

## **7 Felhasználói útmutató**

Tisztelt Felhasználó!

Ezen program lehetőséget nyújt számára, hogy lehető legkézenfekvőbb módon leegyszerűsítse a dolgozók hétköznapi, munkához harmadik kezét nyújtson vagy akár segítse a nyelvtanulását. Elkészítheti a saját szótárát az ön által használt szavak által ezáltal is a rohanó pillanatok közepette, helyesírási pontosságot tud szolgáltatni. Munkahelyén elvárt szaktudását és precizitását legfőképp a szakszavak használata tükrözi. Megeshet bárkivel, hogy épp félregépel pár gombot viszont ezesetben még akár ez is elkerülhető. Egy új nyelvnél természetesen nem tudhatunk folyékonyan beszélni, főképp nem írásban. Utóbbi esetben viszont jó ha van egy eszközünk ami segít félúton írásközben korrigálni minket a helyes leírásban ezáltal is gyakorolva a nyelvet!

### **7.1 Ismerkedés a felülettel**

Program megnyitása után a képernyője jobb alsó sarkában találja a már működő program ablakát. Ablak felső részén közép-jobb oldalt találja az aktuális kifejezést, ami éppen rendelkezésre áll. Az alatti területen jobb oldalt találhatóak meg szavak listája amiből lehet választani. Tőle balra tudjuk kiválasztani milyen nyelv kifejezéseiből akarunk válogatni. Jobb oldali sávban láthatja milyen lehetőségeket kínál a program a szavakkal kapcsolatban. Gombok alatt követhetjük, milyen aktuális nyelvet használunk a szótárunkból.

### **7.2 Ismerkedés a gombokkal**

Számos funkció érhető el a gombok által, amikkel a fenti sávban megjelent szó/szöveget lehet igénybe venni

Send - Elküldi a legutóbb használt szöveges beviteli mezőre.

Add – Hozzáadja a megjelenített szótárunkhoz

Edit – Módosítja a gomb lenyomás előtti szó/szöveget és a fenti sávra beírt szó/szöveggel kicseréli a szótárunkban miután még egyszer rányomunk a gombra.

Remove – Eltávolítja a szótárunkból.

New Language – Ezáltal egy új ablak jelenik meg ahova beírhatunk egy olyan megnevezést ami számunkra egy új nyelvet fog képviselni az elkövetkezőekben. Add gomb lenyomása után elkezdhetjük felvinni a hozzá kapcsolódó szó/szöveget.

## **8 Felhasznált technológiák alkalmazása az oktatási rendszerben**

Ebben fejezetben leginkább az én véleményem és értékrendem szerinti álláspontom szándékozom kifejezni. Továbbá szeretném hangsúlyozni, minden más vélemény, beleértve azokat, amelyek szöges ellentétje a sajátoméval, elfogadom, tiszteletben tartom és ugyan úgy létjogosultságok van, mint az enyémnek.

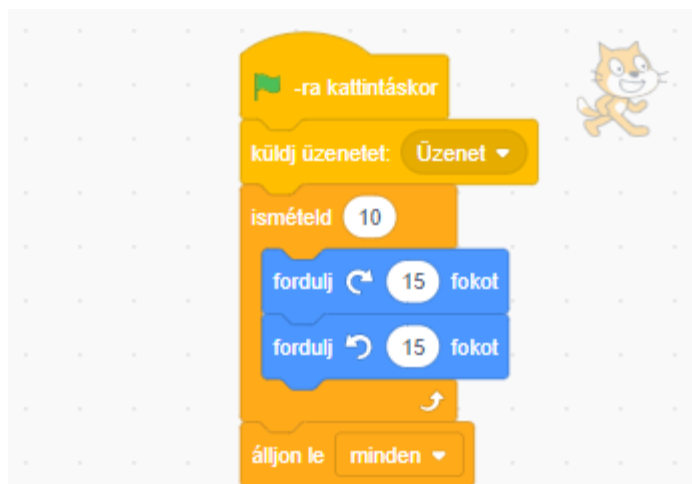
### **8.1 Egyetem első lépései**

Egyetemünk karának szerves részét képezi a gyakorlati programozó ismeretek oktatása. Lépcsőzetesen felépített tárgyrendszer, kezdve a bevezető órákkal, amiknek az lenne a szándékuk, hogy olyanokat is segítsenek megismerkedni és felzárkózni a programozás világával, akik eddig még nem találkoztak vele. Jelenlegi tudásom alapján, ezek a felzárkóztató órák olyan módszereket használnak, amelyek egyszerűek, nem rendelkeznek túlzottan sok lehetőséggel csak az alapokkal, amijenekre ilyen szinten teljes mértékben megfelelőek és sokszor mindezt egy játékos köntösbe van bújtatva. Manapság egyre gyakoribbak az olyan programok, webes applikációk, amelyeknek a célja, hogy bárki számára szórakoztatást nyújtson, miközben célzottan az algoritmusok működését mutatja be, tanítja meg a felhasználót, hogyan kell ezeket használni és a gondolkodásmód elsajátításra sarkall.

Egyik ilyen weboldal a Scratch amiben egy rajzolt macskát tudunk irányítani. Eszköztárunk puzzle alapú, ahol az elemek maguk a parancsok, és funkciójukként megkülönböztetve, formájuk változó. Ez alapján a szintaktikai szabályok, adottak és irányítják a felhasználót, ezáltal megértetve a metódusok megfelelő sorrendjét és hozzájuk szükséges paraméterek használatát. Véleményem szerint a webapplikáció az egyik leghatékonyabb módszer



alkalmazza és így akár egy a szakmában nem jártas számára is tökéletes bevezetést tud nyújtani, hogy megértse vele, a programozáshoz szükséges logikát



23. ábra: Scratch használat közben

## 8.2 Céltudatos szakmai fejlődés

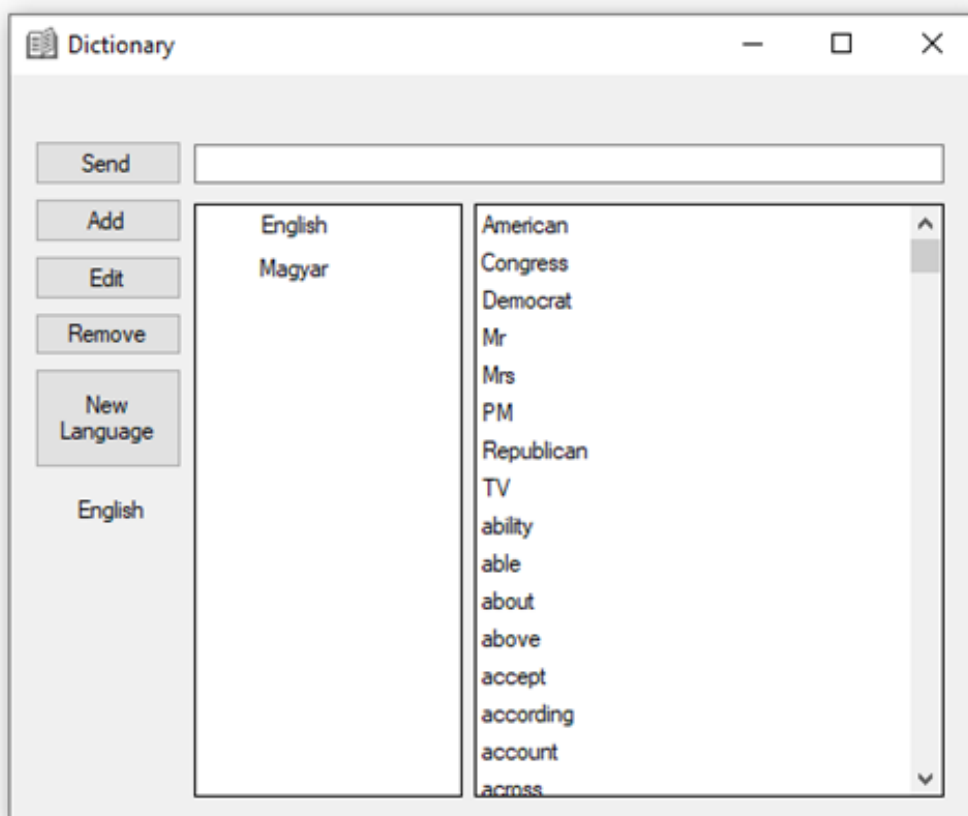
Az idő tájt amikor ezek a bevezető tárgyak nekem voltak aktuálisak, párhuzamosan a fent említett applikáció ismerkedésével a tényleges programozás alapjait, C nyelven sajátítottuk el. Ráépülő tárgyként a későbbiekben C++-t több kurzuson keresztül tanultuk, egyre nagyobb mélységekben. Nyelvek nehézségeit szemelőtt tartva, viszont azt gondolom, hogy a magasabb szintű programozási nyelvekkel határozottan könnyebb a szükséges gyakorlatokat elsajátítani kezdetben. Egy friss hallgató számára nehezebbnek érződik megérteni C nyelven a memóriakezelést miközben az ahhoz szükséges elmélettel, jobb esetben is, akkor találkozik, miközben még az algoritmusok felépítését gyakorolja. Ezen túl lehetőségünk volt Java és C# nyelven is tanulni, viszont ebből csak a Java első kurzusa volt csak kötelező, többi már csak választható gyakorlati tárgynak számított. Tekintve, hogy két teljesen különböző programozási nyelvről beszélünk, érthető, hogy nem érdemes egyszerre mind a kettőnek kötelezően elvégezhetőnek lennie. Mindazonáltal, érdemes lenne megfontolni hogy ezen kurzusokra több ráépülő tárgyat létrehozni hosszútávon, esetlegesen szakirányoknak megfelelően megfelelően felépíteni a program nyelvi és technológiai oktatást.

A C# form alapja lehetőséget biztosít egy egyszerű kezelőfelület létrehozásban ahol könnyedén módosíthatók az attribútumok, ezáltal is az adatok közti kapcsolat is érthetőbbé válik, miközben az elemekhez szükséges funkciókat pedig egy-egy egyszerűbb függvény létrehozásával tudja gyakorolni, hogyan is kell felépíteni és típusok közti különbséget megtapasztalni. Eközben maga a form segíti a hallgatót vizuálisan követni a módosításokat és

az elemek szerinti program építést. Legtöbb IDE már lehetőséget nyújt hogy bővítményeket csatoljunk a projectünkhöz így annak lehetőségeit egyszerűbben tudjuk tágítani. Ezeket a pluginokat sokszor tapasztalt szakmabeli fejlesztők, maguk készítik hogy segíthesse a szakma munkáját vele. Használatukhoz viszont sokszor szükség van a tapasztalatra, hogy könnyű szerrel kiismerhessük, hogyan tudjuk használatba venni munkák során.

Az egyetemen ezek a nyelvek és technológiákról szóló kurzusok elvégzése után szükséges hogy a hallgató hosszútávon is képezze magát. Erre való iránymutatás hiányosnak érződik a tanmenet alapján. A tanulás módszertan és ahhoz szükséges mentális hozzáálláshoz köthető iránymutatás támogatását nem kapják meg, a szerencsésebbek akik komolyabb szakmabeli munkát tudnak végezni cégek alatt, van esélyük a folyamatos szakmai fejlődésre. Az én gondolatom eziránt affelé irányulna, hogy azon oldalak amik leginkább elterjedtek, pld.: Udemy, érdemes lenne külön kurzust hozzá kötni ahol a hallgató kötelezően egy szakmabeli kurzust kelljen elvégeznie és arról beszámolót tartani vagy maga a záró projectjét bemutatni. Így meglehetne alapozni azt hogy hosszútávon az illető képes legyen képeznie magát.

A szakomon leginkább jellemző, hogy minél több tényleges gyakorlati tapasztalathoz



24. ábra: Jelenlegi szótár kinézet úgy mint C# form kinézet

jusson a hallgató. Ez megmutatkozik abban, hogy az általános 320 gyakorlatban eltöltött óra

helyett 500 órát kell munkavégzéssel töltenünk a szakmát képviselve. Számomra ez idő rengeteg lehetőséget biztosított arra, hogy a tudásomhoz kellő magabiztosságot párosítsak és alázatot gyakoroljak a tudásom tágításához.

## Irodalomjegyzék

- [1] <https://blog.oup.com/2007/11/spellchecker/> (letöltés dátuma: 2021. 04 16) When Spellcheckers Attack: Perils of the Cupertino Effect.. | OUPblog:
- [2] <https://www.cnet.com/tech/services-and-software/googles-chastity-belt-too-tight/#The%20Scunthorpe%20problem> (letöltés dátuma: 2021. 04 16) Google's chastity belt too tight | CNET
- [3] <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history> (letöltés dátuma: 2021.04.16) The history of C# - C# Guide | Microsoft Docs
- [4] <https://medium.com/@ByteScout/c-a-brief-history-of-microsofts-premier-language-e4c540c85ddd> (letöltés dátuma: 2021. 04 16) C#: A Brief History of Microsoft's Premier Language | by ByteScout | Medium
- [5] <https://www.ecma-international.org/publications-and-standards/standards/ecma-334/> (letöltés dátuma: 2021. 04 16) ECMA-334 | Ecma International (ecma-international.org)
- [6] <https://www.tiobe.com/tiobe-index/> (letöltés dátuma: 2021. 04 16) index | TIOBE - The Software Quality Company
- [7] <http://www.csc.villanova.edu/~mdamian/threads/posixthreads.html> (letöltés dátuma: 2021. 04 16) Multi-Threaded Programming With POSIX Threads (villanova.edu)
- [8] <https://docs.microsoft.com/en-us/dotnet/standard/native-interop/pinvoke> (letöltés dátuma: 2021. 04 16) Platform Invoke (P/Invoke) | Microsoft Docs
- [9] [https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/ms644985\(v=vs.85\)](https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/ms644985(v=vs.85)) (letöltés dátuma: 2021. 04 16) LowLevelKeyboardProc callback function (Windows) | Microsoft Docs
- [10] <https://docs.microsoft.com/ru-ru/windows/win32/api/winuser/nf-winuser-setwindowshookexa?redirectedfrom=MSDN> (letöltés dátuma: 2021. 04 16) SetWindowsHookExA function (winuser.h) - Win32 apps | Microsoft Docs
- [11] <https://docs.microsoft.com/ru-ru/windows/win32/api/winuser/nf-winuser-callnexthookex?redirectedfrom=MSDN> (letöltés dátuma: 2021. 04 16) CallNextHookEx function (winuser.h) - Win32 apps | Microsoft Docs
- [12] <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-unhookwindowshookex> (letöltés dátuma: 2021. 04 16) UnhookWindowsHookEx function (winuser.h) - Win32 apps | Microsoft Docs
- [13] <https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.keys?view=net-5.0> (letöltés dátuma: 2021. 04 16) Keys Enum (System.Windows.Forms) | Microsoft Docs

[14] <https://www.phraseexpander.com/medical-doctors/> (letöltés dátuma: 2021. 12 06) Text expander for medical transcription and Medical Doctors (M.D.) (phraseexpander.com)

## Mellékletek

Mellékletek mappa tartalma:

Ábrák mappa tartalmazza a szakdolgozatban szereplő képet és ábrát:

1. ábra Hétköznapi esetekben elforduló félrekorrigálás.png
2. ábra Működés fejlesztési terve.png
3. ábra Kezdeti karakter regisztráló form.png
4. ábra Kezdeti vezérlő form.png
5. ábra Funkciók Fejlesztési terve.png
6. ábra TIOBE Programming Community Index grafikonja a leggyakrabban használt kódolási nyelvekről.png
7. ábra Thread működési felépítése.png
8. ábra DLL importálás.png
9. ábra SetHook megvalósítása.png
10. ábra Regisztrált karakter feldolgozás menete.png
11. ábra Futás közbeni felület.png
12. ábra Programhoz tartozó SQL Adatbázis.png
13. ábra Nyelv kiválasztó SQL lekérdezés.png
14. ábra Hozzáadó SQL lekérdezés.png
15. ábra Felülíró SQL lekérdezés.png
16. ábra Törölő SQL lekérdezés.png
17. ábra New Language form.png
18. ábra Send.png
19. ábra Valós idejű kereső SQL lekérdezés.png
20. ábra Google Autocomplete funkcióval működő keresője.png
21. ábra Jelenlegi szótár kinézet.png
22. ábra Sablon kinézet.png
23. ábra Scratch használat közben.png
24. ábra Jelenlegi szótár kinézet – úgy mint C# form kinézet.png

Irodalomjegyzék mappa tartalmazza a felhasznált irodalom weboldalainak lementett változatait

C#\_ A Brief History of Microsoft s Premier Language \_ by ByteScout \_ Medium.mhtml

CallNextHookEx function (winuser.h) - Win32 apps \_ Microsoft Docs.mhtml

ECMA-334 - Ecma International.mhtml

Google's chastity belt too tight - CNET.mhtml

index \_ TIOBE - The Software Quality Company.mhtml

Keys Enum (System.Windows.Forms) \_ Microsoft Docs.mhtml

LowLevelKeyboardProc callback function (Windows) \_ Microsoft Docs.mhtml

Multi-Threaded Programming With POSIX Threads.mhtml

Platform Invoke (P\_Invoke) \_ Microsoft Docs.mhtml

SetWindowsHookExA function (winuser.h) - Win32 apps \_ Microsoft Docs.mhtml

Text expander for medical transcription and Medical Doctors (M.D.).mhtml

The history of C# - C# Guide \_ Microsoft Docs.mhtml

UnhookWindowsHookEx function (winuser.h) - Win32 apps \_ Microsoft Docs.mhtml

When Spellcheckers Attack\_ Perils of the Cupertino Effect \_ OUPblog.mhtml

Program mappa tartalmazza a program tömörített mappáját illetve a mappáját is:

gbc6rq\_dictionary.zip

gbc6rq\_dictionary mappa tartalmazza:

gbc6rq\_dictionary.exe program parancsiconját

Debug mappát ami tartalmazza a programhoz szükséges fileokat és mappákat

database.db

EntityFramework.dll

EntityFramework.SqlServer.dll

EntityFramework.SqlServer.xml

EntityFramework.xml

gbc6rq\_dictionary.exe

gbc6rq\_dictionary.exe.config

gbc6rq\_dictionary.pdb

System.Data.SQLite.dll

System.Data.SQLite.EF6.dll

System.Data.SQLite.Linq.dll

System.Data.SQLite.xml

x64 mappát ami tartalmazza

SQLite.Interop.dll

x86 mappát ami tartalmazza

SQLite.Interop.dll

Táblázat mappa tartalmazza a szakdolgozatban szereplő táblázat képét

1. Táblázat C# verziói fejlődése.png

Tartalmazza a Témakiírásról szóló képet ami a dokumentumban is szerepel

Témakiírás.jpg