ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

**NEUMANN JÁNOS
INFORMATIKAI KAR**

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Nagy Noel**

2021 Hallgató törzskönyvi száma: **T/006324/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Nagy Noel**
Törzskönyvi száma: T/006324/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Emberi reakción alapuló, adaptív pontozási rendszer létrehozása webalkalmazáshoz
Creating an adaptive scoring system based on human reaction for web application

Intézményi konzulens: Sipos Miklós László
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és –fejlesztés specializáció

A feladat

A feladat egy emberi reakciót vizsgáló, és elemző webalkalmazás készítése, mely egy adaptív pontozási rendszer alapján elemzi a felhasználó teljesítményét. Vizsgálja meg, hogy milyen technológiák szükségesek egy webes rendszer és annak háttérben zajló műveleteinek készítéséhez, és elemezze a fentebb említetteket mind negatív, mind pozitív oldalról. Az adott technológia melletti döntését támassza alá. Az alkalmazást csak regisztrációval lehessen használni, az adaptív pontozás végett. A rendszer legyen képes két felhasználó szimultán való együttes kielégítésére, melyben az egyik felhasználó teljesítményét a másik is látja, miközben az adott feladatát végzi el. Az alkalmazás biztosítson lehetőséget arra a felhasználóknak, hogy egyénileg is tudják használni a rendszert, valamint önmaguk eredményeit és azoknak elemzéseit is el tudják érni. A webes felület Angular használatával készüljön el. Miután az egyes felhasználók által eddigi elért eredmények nem változtathatóak, végezzen elemzést a tesztelési lehetőségekről és készítse is el az egyes teszt típusokat és teszteseteket.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- magas szintű szerkezeti mintát az alkalmazásról,
- az egyes funkciók megvalósításához felhasználható lehetséges technológiákat,
- az elkészített rendszer tervét, blokkdiagramját,
- a megvalósítás menetét, felmerülő problémákat és az azokra adott megoldást,
- a tesztelési tervet, a tesztelés módszertanát és a tesztelés eredményeit,
- az elkészült szoftver felhasználási lehetőségeit.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * PH. 1. Budapest

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

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

Budapest, 2022.05.10

Nagy Noel

.....

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
Neumann Faculty of Informatics

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun kód:	Tagozat:
NAGY NOEL	[redacted]	NAPPALI

Telefon: Levelezési cím (pl: lakcím):

[redacted]

Szakdolgozat címe magyarul:

EMBERI REAKCIÓN ALAPULÓ, ADAPTÍV PONTOZÁSI RENDSZER LÉTREHOZÁSA WEBALKALMAZÁSHOZ

Szakdolgozat címe angolul:

CREATING AN ADAPTIVE SCORING SYSTEM BASED ON HUMAN REACTION FOR WEB APPLICATION

Intézményi konzulens:	Külső konzulens:
SIPOS MIKLÓS LÁSZLÓ	---

Alk.	Dátum	Tartalom	Aláírás
1.	2021.09.10.	Félév teendőinek, menetrendjének és mérföldköveinek ismertetése.	
2.	2021.10.18.	Hasonló rendszerek elemzése.	
3.	2021.11.22.	Irodalomkutatás, annak összegzése valamint konzekvenciák meghatározása.	
4.	2021.12.06.	Tervezés, követelmények specifikációja. Véglegesítés. Leadással kapcsolatos teendők egyeztetése.	

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

A hallgató a Szakdolgozat I. (BSc) tantárgy követelményét teljesítette, beszámolóra bocsátható.

Budapest, 2021.12.06.

Sipos Miklós

intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: NAGY NOEL

Neptun Kód: [redacted]

Tagozat: NAPPALI TAGOZAT

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:

EMBERI REAKCIÓN ALAPULÓ, ADAPTÍV PONTOZÁSI RENDSZER LÉTREHOZÁSA WEBALKALMAZÁSHOZ

Szakdolgozat címe angolul:

CREATING AN ADAPTIVE SCORING SYSTEM BASED ON HUMAN REACTION FOR WEB APPLICATION

Intézményi konzulens: SIPOS MIKLÓS LÁSZLÓ

Külső konzulens: -

Alk.	Dátum	Tartalom	Aláírás
1.	2022.03.04.	**Fejlesztés tervezett menetének revíziója, szakmai egyeztetés, fejlesztési teendők megkezdése.**	
2.	2022.04.01.	**Fejlesztés állapotának és haladásának áttekintése. Technikai problémák és módosítások egyeztetése.**	
3.	2022.04.15.	**Felmerülő problémák egyeztetése és megoldása. Tesztelési fázis megindítása.**	
4.	2022.05.06.	**Dokumentáció véglegesítése. Leadással kapcsolatos teendők egyeztetése.**	

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

A hallgató a Szakdolgozat II. tantárgy követelményét teljesítette, védésre bocsátható.

.....

Intézményi konzulens

Budapest, 2022.05.10.

Absztrakt

A jelen korban kijelenthető az emberekről, hogy szeretik tudni önmagukról, hogy milyenek a képességeik. Ezekre számos teszt és feladat létezik, melyek egy sablon eredményt adnak vissza a felhasználónak, mellyel a probléma kézenfekvő: nem egyénileg nézi az adott egyént, ezáltal nem is lehetne releváns az, hogy milyen eredmény jön ki a végén. Szakdolgozatom célja, egy olyan webalkalmazás elkészítése, mely, mivel felhasználói fiókhoz kötött a használata, a felhasználó eddigi eredményeit is számításba véve használ egy adaptív pontozási rendszert a feladatoknál. Az emberről köztudott az a tény, hogy versenyhelyzetben akár szignifikánsabb mértékben jobban tudnak teljesíteni, ezért az alkalmazásban egy olyan lehetőséget tervezek megvalósítani, ahol két felhasználó együttesen, valójában egymás-ellen versengve tudják elvégezni az adott feladatot, miközben a másik előrehaladottságát is érzékelik. Az adaptív pontozás kezdetben minden felhasználónál ugyanúgy indul, viszont az adott felhasználótól függ, hogy milyen mértékben változik meg annak súlypontja, melyekre befolyással van a helyes-illetve helytelen válaszok száma, az egyes feladatoknál eltöltött idő, valamint az adott szavak nehézségi szintje. Ezeket összegezve, egy egyéni képlet segítségével tárolódik el egy belső pontozás a felhasználóhoz. Az egyjátékos mód alap ötletén túl, a kétjátékos módban az ellenfél előrehaladottsága mellett a képernyőn az is kijelzésre kerül a felhasználó számára, hogy a másik játékos milyen szavakat teljesített már sikeresen. A felhasználónak a korábbi eredményeiről számos opció keretében van lehetősége tájékozódni, melyek mindegyike - egy kivételével - egy diagram segítségével tájékoztatja az elért eredmények részleteiről. A kifejlesztett rendszer részeinek működését részenként teszteltem, és kijelenthetem, hogy a különböző részei a rendszernek együttesen működnek az elvártak szerint. Összegzésül elmondható, hogy a specifikációban leírtak nagy részét sikerült megvalósítani, és amellett hogy az általam fejlesztett rendszer működőképes, további fejlesztésekre is alkalmas.

Abstract

Nowadays it is safe to say that people tend to know how good their skills are. For these purposes there are a lot of options out there, which gives the user a general result, but the problem is given: the user is not looked as an individual, but as a user, therefore the result should not be relevant. The goal of my thesis is to create a webapplication, that storages the users previous results, because the usage is tied to registration, and uses these results with the adaptive scoring system at each task. It is well known about man that they can perform significantly better under a competitive situation, so I am planning accomplishing an option, that allows two users to fulfill their tasks while competing against each other with the visibility of the other's progress bar. The adaptive scoring system is the same for everyone at the beginning, but it depends on the user, that how much the system evolves, influenced by the number of right-given answers, the time that was spent at each tasks and the task's difficulty level. Summarizing these, with a unique formula the wepapp stores an inner scoring point for the user. Beyond the basic idea of single player mode, in the two-player mode, in addition to the opponent's progress, the screen will also show the user what words the other player has already successfully completed. The user has the opportunity to find out about their previous results under a number of options, all of which - with the exception of one - provide information on the details of the results achieved using a chart. I tested the operation of the parts of the developed system in parts, and I can state that the different parts of the system work together as expected. In conclusion, most of what is described in the specification has been achieved, and in addition to the fact that the system I have developed is functional, it is also suitable for further developments.

Tartalomjegyzék

1. Bevezetés

A világon számos helyen és módszerrel elérhető egy teszt, vagy egy feladatsor, mely megmondja a feladatot kitöltőről, hogy mennyire okos, milyenek a képességei. Legyen szó egyszerű weboldalról ahol képeket kell kiválasztani a feladat szerint, vagy akár egy komplexebb tesztről, a végeredmény a felhasználó számára egy visszajelzés, hogy a rendszer szerint mennyire ért hozzá. Egy szót legépelni nem nagy nehézség. Mivel a billentyűzeten kiolvasható, hogy melyik, melyik betűt jelöli, akár egy ujjal is le lehet írni bármilyen szót. Viszont ahogy az ember egyre jobban hozzászokik a gépeléshez, annál gyorsabban és gördülékenyebben tud leírni egy-egy szót. De mitől is függ az, hogy egy szót mennyire nehéz leírni? Van-e ennek a tanuláshoz köze? Számításba lehetne venni az egybe-illetve külön írt szavakat, bár ezek az angol nyelvben nem tesznek számot tevő különbséget annyira, mint a magyar nyelvben. Szakdolgozatomban egy saját algoritmust/képletet szeretnék arra használni, hogy az egyes szavaknak a nehézségi szintjét meghatározzam. Bele kell számolni az időt, hogy a felhasználó a képernyőn látható szót, mennyi idő alatt tudja leírni. Ezt nem elég csak másodpercre pontosan mérni, mert a nehézségi szint meghatározásánál több, mint egy befolyásoló tényezőt kell figyelembe venni. A második, mely nagyobb súllyal rendelkezik, az az adott szó hosszúsága, hiszen adott a probléma, minél hosszabb a szó, annál tovább tart azt leírni. De nem csak ez számít akkor, amikor le akarunk gépelni egy szót. Ugyanis észre kell venni azt a tényt, hogy számít, hogy az adott szóban, milyen betűk szerepelnek. Amikor papíron írunk le egy szót, nincs hatással az előző betű a következőre, mert egy kézzel írunk, folyamatosan, betű után betűt. Amikor gépelünk, akár mind a tíz ujjunkat használhatjuk, lényegében szinte párhuzamosan tudunk írni. Idővel, miután az ember hozzászokott a gépeléshez, a keze automatikusan egy adott pozícióban helyezkedik el a billentyűzet felett, az egyes ujjai, melyekkel gépelni szokott, a billentyűzet egy részét fedik le, és amely betűk hozzá esnek, azokat azzal az ujjal nyomja le. Ha egy szóban két betű szerepel, melyek a billentyűzet kiosztásán egymástól messze helyezkednek el, akkor könnyen és gördülékenyen, egymás után, bármilyen probléma nélkül leírhatóak. Abban az esetben viszont, amikor egy szó minden betűje, nagyjából azonos területen helyezkedik el, nem lehet olyan szimultán megoldani a gépelését, például az angol „sad” szó (jelentése: szomorú), három betűből áll, mindegyike egymás mellett helyezkedik el, ezért ezt szeretném nehezebb szónak jelölni, mint például egy „nod” (jelentése: bólintás) szót, melynek karakterei egymástól eltérő területen szerepelnek a kiosztáson.

Célom, hogy a szakdolgozatom befejezésére egy olyan webalkalmazást tudjak létrehozni, melyben két felhasználó egymás ellen versengve is tudja használni és megmérni tudását, valamint az egyes felhasználók önmagukban is tudják fejleszteni gépelési képességeiket, figyelembe véve az eddigi teljesítményüket és a feladatokat.

2. Irodalomkutatás

2.1 Hasonló rendszerek elemzése és bemutatása

A szakdolgozatom témájával egyező irányú webalkalmazásokat és rendszereket, egy gondolatmenet szempontjai alapján szeretném elemezni. Azt, hogy az emberi reakciót és a tudását elemezzük, nagyon sok féle képpen tehetjük. Lehetőségünk van csak gyorsaságot elemezni, valamint készségeket is. Fontos azonban belekalkulálni, hogy nem minden ember indul egyenlő eséllyel egy adott feladatnál, így fontos tud lenni az előmenetel, valamint, hogy a feladat személyre szabott legyen. Két kategóriára tudnám ezeket bontani leginkább.

2.1.1 Emlékezet nélküli rendszerek

Azok a lehetőségek, melyben a felhasználónak nem kell semmilyen előzetes tudással rendelkezni, és levegőből kapott ötletben hajt végre egy feladatot, amivel a reakcióját akarja elemezni, a tudását feltérképezni, emlékezet nélküli rendszernek tekintem. Ilyenkor a rendszer nem azonosítja be az adott felhasználót, nem szab köré egyedi azonosítást, feladatot, sem pontszámot. Minden felhasználó egy ilyen rendszernél egyenlő eséllyel indul, minden alkalommal randomizáltan történik a feladat nehézsége, valamint az esetleges ellenfelek kiválasztása is. Nincs mérvadó pontrendszer, amivel az azonos előzetes tudás után megszerzett pontú emberek kerülnének egymás ellen egy megméredtetésben.

Ebbe a kategóriába esik az, melyben egy webalkalmazáson egymás ellen versenyezve lehet a szavakat leírni gyorsaság szerint. Egy népszerű webalkalmazást választottam arra, hogy ezt a kategóriát be tudjam mutatni, mely a Typeracer webalkalmazás.

A Typeracer széles körben elterjedt oldal azon közösség körében, akik az átlagosnál több időt töltenek a billentyűzet használatával. Az oldalt regisztráció mentesen is lehet használni, viszont ha regisztrálunk, akkor több funckiót is el tudunk érni, melyeket csak vendégfelhasználóként nem is tudhatunk, hogy létezik. Az oldal fő monumentuma, hogy

1. Ábra: A Typeracer oldal használat közben

be lehet lépni egy versenybe, ahol összesen négy másik felhasználóval indul el a játék. Amikor egy játék elindult, azaz meglett a kellő létszám hozzá, az egyes játékosokat reprezentáló kocsik is megjelennek a képernyőn, valamint az is, hogy milyen szöveget kell legépelnie a felhasználóknak. A szöveget karakterre pontosan kell leírni. Amikor az egymás után következő karakterekben feltűnik az első elírás, hibát jelez a webalkalmazás, és nem engedi továbbírni a szöveget, ameddig azt az elrontott karaktert ki nem javítjuk. Ahogy a képen is látható, a négy játékos közül három regisztráció nélkül használja az oldalt, és egyedül a narancssárgával jelölt játékos tudja az eredményeit visszanézni. A jobb oldalán pedig minden játékosnak az eddig legépelt szavakból alkotott wpm[1] látható, mely folyamatosan változik az idő, valamint a kész szavak számának függvényében. Amikor egy felhasználó végzett a beírandó szöveg legépelésével, erről egy összegzést kap, melyben szerepel: a pontossága, a wpm értéke, valamint az eltelt idő és egy magyarázat arra, hogy mi a legépelt szöveg eredete. Technológiát tekintve a Google által kifejlesztett OpenSocial API[2]-t, valamint a Google Web Toolkit[3]-et. Ezt a mindennapjainkban egy átlag felhasználó nem veszi észre, de amikor csak egy üzenetet küldünk el pl. Facebook-on, akkor is az API szolgáltatását használjuk.

A Typerush nevezetű oldal megszólalásig megegyezik a fentebb említett webhellyel, egy különbséget eltekintve: itt a leütött rossz billentyűt nem kell kitörölni amikor tovább akarunk haladni. A fejlesztők itt egy kicsit nagyobb felhasználói interfészt hoztak létre, és az egyes játékok után nyereményeket lehet kapni, valamint pénzt gyűjteni, melyeket elköltve új autót - ami a játékost reprezentálja használat közben - lehet vásárolni. Az oldalon van egy Stats[4] lehetőség, mely az eddigi pályafutásunkról ad egy összegző visszajelzést, ami már hasonlít a szakdolgozatom ötletéhez, de ezek a korábbi eredmények nincsenek befolyással a következő alkalomra, ezáltal csak korábbi eredményként tudnék rá legjobban hivatkozni.

[1] „words per minute” fordítás: percenkénti szavak száma
[2] Alkalmazás interfész, ami egy szoftver közvetítő, mely megengedi két alkalmazásnak a kommunikációt
[3] Egy nyílt hozzáférésű fejlesztői eszköztár, melynek feladata, hogy Java kódból állít elő JavaScript, illetve HTML kódot
[4] „statistics” fordítás: statisztika, a korábbi eredmények

2.1.2 Emlékezettel rendelkező rendszerek

Ezt a kategóriát két olyan oldallal szemléltetem, ahol a tudáson van a hangsúly, ahol alkalmazva van – tanulási szemszögből – az emlékezés, azaz a használhatóság egyik alapelve, hogy egymásra épülés van. A leginkább ilyen közismert oldal, mely valamelyest is rendelkezik előismerettel az egyes felhasználóról amikor az újra használni kezdi az oldalt, az a Duolingo. Mely egy nyelvtanító program lényegében. Széles választékkal rendelkezik az oldal, mely nem csak a magyar emberek között terjedt el, hiszen nem csak magyarra értelmezhető az adott tanulni kívánt nyelv. Az oldal fő célja, hogy a felhasználó kiválaszt egy idegen-nyelvet, melyet szeretne megtanulni, és a már előre elkészített struktúra alapján felépített tananyagon tud végighaladni, melyben az elején egy-egy szó jelentését kell megfejteni a lehetőségek közül. Ahogy a felhasználó egyre előrébb jut, annál nehezebb szavakat fog kapni, és a feladatok is bonyolultabbak lesznek. Kis

2. Ábra: Duolingo kezdő feladat példa

előrehaladás után már nem csak olvasva kell felismerni a megtanult szavakat, hanem hallás alapján, valamint a felhasználó kiejtését is elemzi az oldal, és annak megfelelően elfogadja vagy elutasítja.

A felépített tanangyag egymásra alapoz, ezért a legelején amikor egy új felhasználó kezdi el használni az oldalt, egy idegen nyelv legalapvetőbb és könnyebben megérhető szavaival illetve kifejezéseivel találja szemben magát. Egy előre felállított, jól megtervezett és tanárok által jóváhagyott oktatási rendszert használ a Duolingo. Az oldal legelőször 2011 novemberében indult útjára, pontosabban ennek a béta verziója, melyre több mint 300 ezren jelentkeztek, és ennek hatására kicsivel több mint fél évre rá már több mint 10 millió ember használta az oldalt világszerte. A kezdetektől fogva a Doulingo az Amazonnak a webszerverein van futtatva, az adattárolásra és a felhasználói szókincshez SQL-t illetve DynamoDB-t használ. Az oldal backend részen Pythont használt, de a fejlesztők nemrég átváltottak a Scala használatára hosszú évek után, ennek pedig a fő oka a gyorsaság hiánya volt. A Scala egy statikusan beírt nyelv, mely széleskörben elterjedt a nagy adathasználatú applikációk körében.

A másik olyan oldal, melyet ebbe a kategóriába tudnék besorolni az a Honfoglaló. Magyarországon hatalmas népszerűségnek örvendett, még egy TV műsort is készítettek ennek az oldalnak az alapfelépítését és ötletét követve. A játék menete abból állt, hogy Magyarország térképe volt a játéktér, azon belül a megyék voltak az egyes elfoglalható lehetőségek. Minden játék akkor indult el, amikor három játékos is becsatlakozott egy meccshez. A Honfoglalóban körönként új és új, különböző témákból érkeztek a kérdések, melyek vagy felelet választó kérdések voltak, vagy pedig a billentyűzet segítségével egy végső számjegyet kell beírni a kérdés szövegének megfelelően. Miután mind a 19 megye kiosztásra került, elkezdődött a támadás rész, ahol mindenki a saját körében meg tudott támadni egy általa választott megyét a másik játékostól, és amennyiben a kérdésre ő adott helyes választ, vagy gyorsabban írta be a megoldást, elnyerte a megyét és a pontszámát.

Kezdetben, amikor egy játékos megnyert egy mérkőzést, azaz vagy elfoglalta egész Magyarország területét, vagy pontszámügyileg övé volt a legtöbb, egy csillagot kapott a saját profiljában. Amennyiben a következőt is megnyerte, kapott mégegyet, viszont ha elveszítette, akkor elvesztett egyet, viszont ha még nem volt neki és úgy veszített, kapott egy szimbólumot, mely a mínuszt jelentette. Amennyiben egy nap egy felhasználó össze tudott gyűjteni öt csillagot, megkapott egy koronát, mely a csillaggal ellentétben, a nap végén nem nullázódott le, és ez a korona jelképezte az adott tudásszintet a játékban, ez alapján kerültek össze az ellenfelek.

Az oldal megszűnése előtt, a fejlesztők sok új és kedvező funkciót behoztak a felhasználói élmény növelése érdekében, ilyen volt például a felező segítség, mely a felelet választás típusú kérdésnél, a négy lehetőséget lefelezte, ezzel növelve a megoldás eltalálásának esélyét. Ezenfelül, a koronás szintbeli rendszer megszűnt, hiszen voltak akik nagyon jó játékosok voltak, viszont nem tudtak öt meccset lejátszani egy nap, hogy magasabb szintre jussanak, ezért bevezették a tapasztali ponton alapuló szintmeghatározást. Itt a lejátszott meccsek alapján kaptak tapasztalati pontokat a játékosok, és nőtt a szintjük.

2.1.3 Összegzés

Az emlékezet birtoklásán kívül persze több csoportba is bonthatóak lennének a megvizsgált rendszerek, akár nézhetnénk azt is, hogy tudást mérnek, vagy csak gyorsaságot, esetlegesen mindkettőt. Az interneten számos olyan oldal is van, mely a szavak begépelése terén működik, viszont mindegyik csak az adott pillantra fókuszál, és egy-egy konkrét mondatot, mely általában egy idézet, ad feladatul a felhasználónak. Az alkalmazásom, melyet a szakdolgozatomként tervezek megvalósítani a feldolgozott rendszerek egyes egyedi tulajdonságait ötvözné, de megőrizné egyediségét a pontozási rendszer felépítése téren. Egyszerre alkalmazná az emlékezet nélküli rendszereknél bemutatott Typeracer oldal alap felvetését a gyorsaság és szavak számának függvényében, kiegészítve az adaptív pontozással, valamint a nyelvtanító Duolingo azon elemét, hogy a felhasználói korábbi eredményei tárolásra kerülnék, és ezek alapján történik meg a következő feladat kiosztása az adott felhasználóhoz. Ezentúl, a Honfoglalóban ismertetett, hasonló szintbeli ellenfelek kiválasztását, amennyiben ez lehetséges az éppen aktuálisan várakozó játékosoknál.

2.2 Webalkalmazások működése

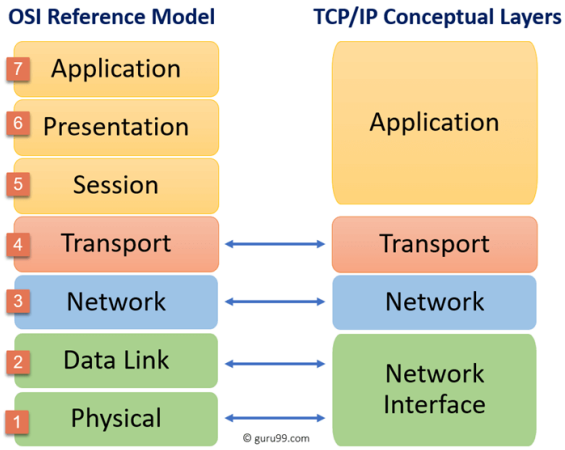
Az emberek egy átlagos nap rengeteg weboldalt keresnek fel információért, tájékozódásért. A legtöbb webalkalmazás megformázva kerül a felhasználó elé. Ezeket a fejlesztők egyedileg alkotják meg, egy úgynevezett CSS[5] segítségével. Ezzel egyedileg beállítható és személyre szabható minden egyes kis része egy weboldalnak. Az oldalon megjelenő adatokat viszont a HTML[6] jeleníti meg.

Azokat számítógépeket, melyek az internethez vannak csatlakozva, klienseknek illetve szervereknek nevezzük. A kliensek a tipikus internetet használók eszközei, például a számítógépe egy embernek ami a Wi-Fi-hez van csatlakozva, melyen van, legalább egy internetet elérő böngésző pl. Google Chrome. A szerverek azok a gépek, melyek weblapokat, oldalakat, applikációkat tárolnak el. Amikor egy kliens eszköz elérést kér egy oldalhoz, annak egy másolata töltődik le a szerverről a kliens gépére, melyet fog megjeleníteni annak böngészője. [1]

De mi is történik pontosan, amikor egy webcímet írunk be a böngészőbe. A böngészőnk eljut egy DNS[7]-hez ami lényegében a telefonkönyve is az internetnek. Ez a DNS szerver felelős azért, hogy a beírt webalkalmazás címéhez tartozó IP címet megtalálja. Ezután a böngésző küld egy HTTP[8] kérést a szerver felé, amivel egy másolatot kér le a webalkalmazásról a kliensnek. Ez a kérés, és minden adatátküldés a kliens és a szerver között TCP/IP[9]-vel történik. A TCP/IP felépítése a rétegződési elven alapul, melyben minden egyes réteg egy jól definiált feladatot végez el, és a rétegek egymás között szolgálatelérési pontokon keresztül kommunikálnak. Minden réteg csak a vele szomszédos réteggel képest kommunkálni, öt ilyen réteg van:

1. Alkalmazási réteg: A felhasználó által indított program és a szállítási réget között teremt kapcsolatot. Ezen réteg protokolljaival az alkalmazások képesek egyeztetni a formátum illetve biztonság, vagy más hálózati igényekről. pl.: HTTP
2. Szállítási réteg: Az alkalmazási rétegtől kapott adat elejére egy úgynevezett fejlécet csatol, mely jelzi, hogy melyik szállítási rétegbeli protokollal küldik az adatot. Biztosítja hogy a felhasználók közötti adatátvitel transzparens legyen, és elvégzi az adatfolyam szegmentálását, illetve deszegmentálását
3. Hálózati réteg: Biztosítja a változó hosszúságú adatok eljutását a küldőtől a célállomásig, ennek módja az IP címzés
4. Adatkapcsolati réteg: Lehetővé teszi két hálózati elem kommunikációját
5. Fizikai réteg: Továbbítja az adatkapcsolati rétegtől kapott kereteket a hálózaton

[5] Cascading Style Sheets, egy stílus leíró nyelv, mely a HTML dokumentumok megjelenését írja le
[6] Hypertext Markup Language, egy leíró nyelv, amit weboldalak készítéséhez fejlesztettek ki [36]
[7] Domain Name System, domain nevek rendszere, azaz egy tartománynévrendszer
[8] HyperText Transfer Protocol, egy információátviteli protokol
[9] Magyarra fordítva az átviteli vezérlő internet protokoll rövidítése



3. Ábra: TCP/IP a régebbi OSI modellel összevetve

Ha a szerver jóváhagyja a kérelmet, küld egy „`200 OK`” üzenetet, mely egy visszaigazoló válasz, miszerint sikeres volt a kérelem, valamint elkezdi elküldeni a weboldalnak a fájljait a böngészőnek, apró darabok sorozataként, melyeknek a neve adatcsomag.

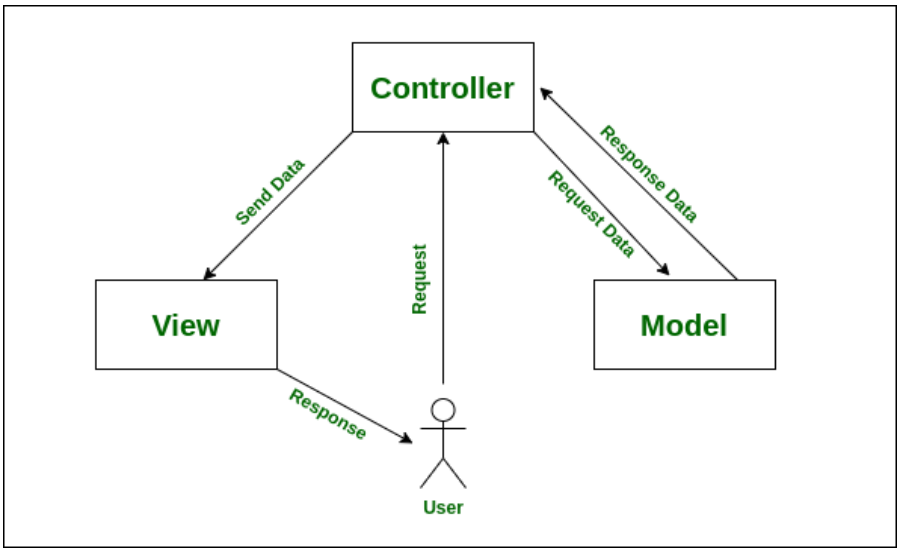
2.3 ASP.NET

Az ASP.NET egy nyílt forráskódú web applikáció keretrendszer, melyet dinamikus és szerveroldali applikációk gyors létrehozásához használhatunk. Osztályok és komponensek együttese, mely MVC[10] tervezési mintát használ. Az ASP.NET arra is felhasználható, hogy HTTP API-kat hozunk létre, melyeket el tudunk érni bármilyen telefonos applikációval, vagy böngészőn alapuló egyoldalas applikációkkal, mint pl.: Angular vagy a React. [2]

Az MVC három főbb rétege a Model, a View és a Controller:

- Model: leginkább az adatokat képviselő réteg és definiálja az applikáció összes objektumának tárolását
- View: a vizuális megjelenítésért felelős, és a leginkább interakcióban levő réteg a felhasználói interfésszel, kezeli a kommunikációt a felhasználó és a controller között
- Controller: másnéven az üzleti logika, mely az „agya” az applikációnak, ez kezeli a kéréseket, a felhasználó outputját átalakítja megfelelő formátummá, és azt továbbítja a megfelelő felhasználó nézethez [3]

[10] Modell-View-Controller, egy tervezési minta mely felhasználói interfészeket, adatokat és üzleti logikát implementál



4. Ábra: MVC működésének bemutatása

Struktúrát, segítő funkciókat, valamint egy applikációkat felépítő keretrendszert biztosít az ASP.NET, mely sok kódírástól tudja megmenteni a fejlesztőt. Ezután a keretrendszerben megírt kód meghívja kezelőket, mely lényegében az applikáció üzleti logikáját képező rétegben levő metódusok. Ez a réteg a magja az applikációnak. Ezen keresztül tudunk interakcióba lépni más szolgáltatásokkal, mint pl.: adatbázisok, távoli API-k de ettől függetlenül, az üzleti logikája az applikációnak nem teltétlenül függ közvetlen az ASP.NET Core-tól. [4]

Az egyik fő indoka annak, amiért széles körben elterjedt lett az ASP.NET, az az, hogy képesek vagyunk fejleszteni és futtatni vele minden platformon. Ahogyan a futtatási sokszínűsége, úgy az „egyszer megírjuk és működik” elve is a siker egyik pontja. Az applikáció IL[11] kóddá áll össze. Ha a célrendszeren fel van telepítve a .NET 5.0, akkor bármilyen platformról származó IL összeállítású kódot tud futtatni.

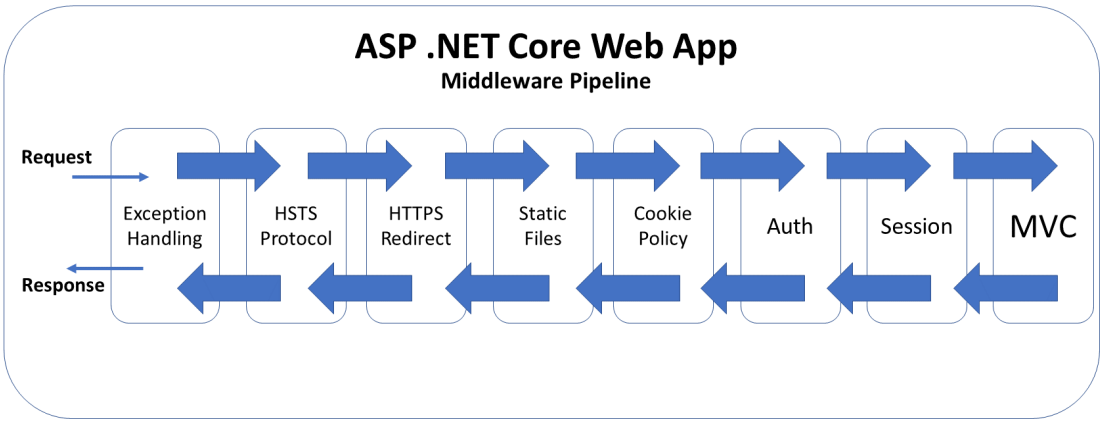
Az ASP.NET-el a szerveren tárolt adatokból állítunk elő HTML oldalakat, amit a kliensek könnyen meg tudnak jeleníteni. A hagyományos HTTP protokollt alkalmazza, mely a legszélesebb körben elterjedt protokoll webapplikációk terén. Az ASP.NET a Microsoft által fejlesztett .NET környezet egy része. Minden objektum egy gyakran használt funkciók egy csoportját valósítja meg, melyek hasznosak dinamikus oldalak fejlesztésénél. ASP 2.0 -tól hat ilyen beépített objektum van: Application, ASPError, Request, Response, Server és Session. Az ASP.NET applikációk egyik előnye, hogy sok nyelven írhatóak pl.: C#, JavaScript. [5]

A felhasználó tevékenységét egy eseményvezérelt modellen alapuló mechanizmus segítségével dolgozzák fel. Ha például a felhasználó egy nyomógombot megnyom, vagy

[11] Intermediate Language: egy objektum orientált programozási nyelv a .NET keretrendszerben

egy szövegbeadási mezőt kitölt, vagy épp egy rádiógomb állapotát megváltoztatja, akkor egy ehhez tartozó eseménykezelő fog lefutni a szerveroldalon. A szerveroldalon ezeket az eseménykezelőket megírhatjuk C# vagy Visual BASIC.NET, akár más .NET alapú nyelveken.

A felhasználói folyamat akkor indul el, amikor elnavigálunk egy oldalra, így a megfelelő URL[12] kiválasztódik a böngészőben. Az URL, vagy a webcím egy hostnévből és egy útból áll. A navigáció küld egy kérést a felhasználó számítógépétől a szerver fele, amelyen az oldal futtatva van, HTTP protokolt alkalmazva. Amikor egy ASP.NET Core webapplikációt készítünk, a böngésző az úgyanúgy, azokat a protokollokat alkalmazva kommunikál az applikációval. Magába foglal mindent ami a szerver oldalon helyezkedik el ahhoz, hogy egy kérést kezelni tudjon, beleértve a kérésnek validációját, a bejelentkezés lebonyolításának részleteit és a HTML fájl generálását. Minden ASP.NET-es applikációnak van egy belső webszervere, ami alapból a Kestrel, ez felelős a nyers kérések fogadásáért és ezekből az adatokból egy belső reprezentáció felépítéséért, mely egy HttpContext objektum. Az applikációnak ezen reprezentációból képesnek kell lennie egy megfelelő választ kreálni a kérésre. Ez akár használhat részleteket és információt a HttpContextben eltároltakból, hogy a választ legenerálja, amihez akár kellhet, hogy egy HTML-t szükséges lérehoznia, ha a „`Belépés megtagadva`” választ kell nyújtania.



5. Ábra: ASP.NET Core applikáció kérés feldolgozás menete

Amikor az applikáció befejezte a kérés feldolgozását, visszaküldi a válaszát a webszervernek. Az ASP.NET webszerver ezt fogja konvertálni egy nyers HTTP válasszá és küldi tovább a hálozatnak, ami majd továbbítja a felhasználó böngészőjének. Az összes különbség a generikus HTTP kéréshez mérten, szerveroldalon található meg.

[12] Uniform Resource Locator, másnéven az oldal webcíme

2.4 JavaScript

A JavaScript 1995-ben jött létre, és eredetileg LiveScript-ként volt elnevezve Brendan Eich, a készítője által, viszont mivel a piacon akkoriban mindenhol csak szinte Java volt megtalálható, ezért marketing szempontok miatt megkapta az új, és azóta is használt JavaScript nevét, függetlenül attól, hogy semmi köze sincs a Java-hoz.

A JavaScript egy könnyű, de elképesztően erős, objektum orientált script nyelv. A JavaScript egy dinamikus nyelv, ahol új tulajdonságokat lehet hozzácsatolni egy objektumhoz. Ezt a ***Prototype*** objektummal tudjuk elérni, ami alapból, minden objektummal és függvénnyel össze van kapcsolódva, ahol a függvénynek a prototype tulajdonsága hozzáférhető és módosítható, valamint az objektumnak a prototype tulajdonsága, másszóval attribútuma nem látható. Ez a Prototype objektum egy speciális típusú felsorolható objektum amelyhez további tulajdonságok kapcsolhatóak, amelyek meg lesznek osztva a konstruktor funkciójának összes példánya között. Ezért is nevezik gyakran nem objektum orientált nyelvnek, hanem Prototípus alapú nyelvnek. [6] Leggyakrabban a böngészőnk használata közben botlunk bele a JavaScript-be, de manapság már a legalapabb applikációktól az ebook-okon fellelhető PDF-ekben is fellelhető a kódja, valamint még a webszerverek maguk is tudnak JavaScript által működni. [7] A JavaScriptben írd kódban minden amit létrehozunk már objektum, vagy éppenséggel átalakul objektummá amikor szükséges. A kód egy külön „`.js`” kiterjesztésű fájlban található meg, vagy a HTML fájlban, ezeket akár a legegyszerűbb szövegszerkeztővel is könnyedén szerkezetni tudjuk. [8]

Mind kliens oldalon, mind szerver oldalon használható nyelv a JavaScript. [9] Mint egy dinamikus programozási nyelv, a JavaScriptet sem kell közvetlen valamilyen összeállító formon keresztül futtatni, amely az emberi olvasásra alkalmas kódot átalakítja olyanná, melyet a böngésző értelmezni tud. A böngésző effektíven olvassa a kódot ahogy mi, és eközben, azaz futás közben értelmezi is. Mivel ez egy dinamikus nyelv, így az oldal melyet ennek segítségével hozunk létre is dinamikus lesz. A statikus oldalakkal ellentétben, itt nem fixált tartalom jelenik meg, hanem változhatnak az adatok, a megjelenés, felhasználótól és annak interakciójától függően. A dinamikus oldalak egyik fő tulajdonsága még, hogy legalább egy adatbázishoz csatlakozva vannak, ahol tárolódnak el a dinamikus adatok, statikus weboldaknál nincs ilyen adatbázis.

A JavaScript egy gyengén típusos nyelv, ami azt jelenti, hogy nem feltétlen kell elmondani, egy változó típusát. Például, ha egy változónak megadjuk az értékként, azt hogy 5, akkor programozottan nem kell meghatároznunk, hogy az a változó egy szám, ezt a JavaScript magának fogja felismerni. De ezt felül is tudjuk írni. [10]

Amikor egy oldal betöltődik, a böngésző létrehoz egy DOM[13]-ot az oldal alapján. A DOM egy programozói interfész webes dokumentumokhoz. Ez reprezentálja az oldalt azért, hogy a programok tudják változtatni a dokumentum struktúráját, stílusát és tartalmát. Csomópontok és objektumok segítségével jeleníti meg a dokumentumot a DOM, így a programozói nyelvek képesek kapcsolatba lépni az oldallal. Egy weboldal valójában egy dokumentum, amit vagy megjeleníteni lehet a böngésző egy ablakában, vagy egy HTML forrásként. Mindkét esetben ugyanaz a dokumentum, viszont a DOM megjelenítés lehetővé teszi annak manipulálását. Mivel ez egy objektum orientált megjelenítése az oldalnak, egy script nyelvvel, mint például a JavaScript, módosítani lehet azt. [11]

Az így megkapott objektum modellel a JavaScript képes arra, hogy egy dinamikus HTML-t hozzon létre. Hiszen, változtatni tudja ennek a HTML-nek az összes elemét, attribútumát, ezeknek kinézetét, stílusát is szabályozni tudja. [12] A DOM nem része a JavaScriptnek, ehelyett egy API-ként használják webalkalmazások felépítéséhez.

A JavaScript rövid időn belül felfejlődött, a callback[14]-ektől a Promise[15]-okig, és az ES2017-től kezdve bevezették az aszinkron JavaScriptet, ami még könnyebb lett az „***async***” valamint „***await***” szintaxissal. Egy aszinkron függvény egy Promise visszatérési értéke van, és amikor egy ilyen függvényt akarunk meghívni, akkor ez a hívás addig várakozni fog, amíg a Promise meg nem lett olda, vagy el nem utasítva. [13]

```
const doSomethingAsync = () => {
  return new Promise(resolve => {
    setTimeout(() => resolve('I did something'), 3000)
  })
}

const doSomething = async () => {
  console.log(await doSomethingAsync())
}
```

6. Ábra: JavaScript-ben aszinkron függvény és annak meghívása

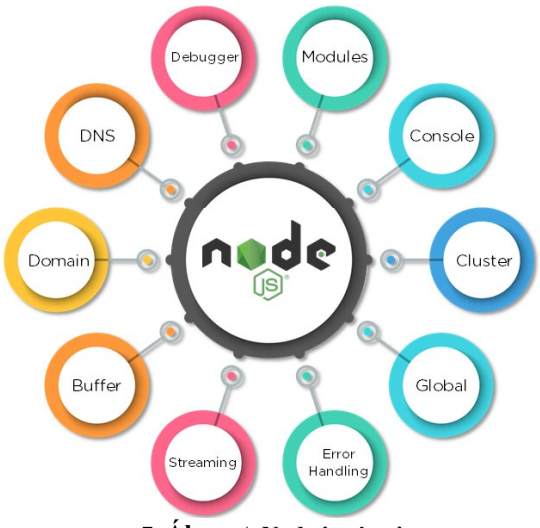
2.5 Node.js

A Node.js egy szerver-oldali platform, mely a Google Chrome JavaScript V8-as Engine-en épült. 2009-ben lett kifejlesztve a legelső verziója Ryan Dahl által azon célból, hogy lehetőséget biztosítson arra, hogy a fejlesztők JavaScript-et tudjanak használni szerver-oldali scriptelésre. Node.js egy nyílt forráskódú, platformfüggetlen JavaScript futásidejű környezet, nem könyvtár, webes applikációk futtatására a kliens böngészőjén kívül. Ezen alkalmazások JavaScript nyelven íródnak, és a Node.js-ben futtatható megannyi operációs rendszeren, pl. Windows, Linux. [14]

[13] Document Object Model: az adatmegjelenítése azon objektumoknak melyek tartalmazzák az oldal tartalmát és szerkezetét.
[14] A callback, magyarul: visszahívás, egy függvény, amelyet argumentumként adnak át egy másik függvénynek [38]
[15] A promise, magyarul: ígéret, egy JavaScript objektum, amely összekapcsolja az előállító kódót és a felhasználó kódót [39]

Számos indok van ami miatt a Node.js elsőszámú válaszások közé került már szoftver architektúrák terén, mint például az ,hogy aszinkron módon működik, valamint eseményvezérelt. A Node.js könyvtárban található összes API aszinkron, azaz nem blokkoló, miszerint sosincs olyan, hogy egy API visszatérési adatra várakozni kelljen.

Így több folyamatot is tud egyidejűleg kezelni, nincs az, hogy a felhasználónak várakoznia kelljen arra, hogy egy előző folyamat, például egy fájlbeolvasás, vagy léptetés véget érjen. Hozzátartozik egy NPM[16] is.



7. Ábra: A Node.js részei

Ahogyan a 7. ábrán is látható, tíz fő része van, melyek együttesen alkotják meg. A modulok, mint a JavaScriptben a könyvtárak, használhatóak a Node.js-ben azzal a céllal, hogy különböző funkciókat tudjunk elérni az applikációban. Három fajta modul található meg a Node.js-ben: Core (mag) modul, Local (helyi) modul és Haramdik féltől származó modul.

- A core modulok tartalmazzák az alapvető, tiszta funkcionalitásait a Node.js-nek. Automatikusan betöltésre kerülnek amikor egy node parancs elindul.
- A local modulok azok, melyek a Node.js applikáción belül lettek létrehozva. Plusz, valamint különböző funkciókat tartalmaz egy külön fájlban és mappában.
- A harmadik féltől származó modulok, melyek már egy előre megírt kódot tartalmaznak, ezeket be lehet importálni az applikációba. [15]

Ha bármelyik ilyen modult alkalmazni szeretnénk, akkor használnunk kell a „***require()***”-t. A Console (konzol) egy olyan modulja, mely egy debugging (hibakeresés) módszert biztosít. A Cluster fogja engedélyezni azt, hogy több szálat futtassunk, mindezt úgy, hogy egyfajta „gyermek folyamatot” hoz létre, mely ugyanazon a szerveren és porton fut,

[16] Node Package Manager: Node csomag kezelő, amely több mint ötvenezer csomagot tartalmaz, melyben bármilyen applikáció fejlesztéshez megtalálható az éppen szükséges csomag, és azonnal telepíthető is.

szimultán. Négyfajta hibatípust különböztetünk meg a hibakezelésen belül, melyek mindegyikét kivételkezeléssel old meg a Node.js. [16]

A Node.js mechanikája ami miatt annyira népszerű, míg más alternatív megvalósítások több szálú feldolgozást alkalmaz, a Node.js megoldja ezt mindösszesen egyetlen szállal. Több szálú feldolgozásnál, egy szálat akkor kezdünk el használni, amikor egy kérés érkezik, amennyiben nincs szabadon elérhető szál, mert mindegyik egy feladattal van elfoglalva, akkor a szervernek várakoznia kell, hogy valamelyik jelenleg használatban levő szál felszabaduljon. Ettől az applikáció lassú lesz és nem hatékony. Ezzel szemben a Node.js egyszálú feldolgozást használ. Ilyenkor egy fő szál van, ami minden kérést feldolgoz, esemény hurkokat használva arra, hogy azokkal legyenek futtatva az alapból blokkoló hatású kérések, ami például a fájlok kiírása és beolvasása, úgy, hogy azok ne legyenek blokkolóak. [17]

2.6 TypeScript

A JavaScript kezdetlegesen a kliens oldalra lett bemutatva, mint egy programozási nyelv. A Node.js fejlesztése viszont megjelölte a JavaScriptet, mint feltörekvő szerveroldali technológia is. Mindazonáltal, ahogy a JavaScript kód növekedett, egyre inkább hajlott arra, hogy nehezen kezelhető legyen, fentartható, újrahasználható. Továbbá, a JavaScript kudarcot kezdett vallani abban, hogy magába foglalja az objektum orientált programozás jellemzőit, az erősen típusosságot, és a fordítási időben történő hibák ellenőrzését, s mindezek megakadályozták a JavaScriptet abban, hogy sikeres legyen továbbra is vállalati szinten, mint egy teljes értékű szerver oldali technológia. Ekkor érkezett meg a TypeScript, hogy ezeket a réseket kitöltse. [18]

A TypeScript egy erősen típusos, objektum orientált futásidejű programozási nyelv. Anders Hejlsbert tervezte meg (aki a C# tervezője is volt) a Microsoftnál. A TypeScript mind egy nyelv, mind pedig eszközök készlete. A JavaScripthez képest leginkább úgy jellemezhető, hogy a TypeScript az JavaScript, plusz néhány további lehetőséggel.

A TypeScript támogatja a definíciókat tartalmazó fájlokat, melyekben megtalálhatóak a típus információk egy már létező JavaScript könyvtárhoz, úgy mint C++ nyelvben a header (fejléc) fájlok is le tudják írni a már létező objektum fájlok struktúráját. Ez engedélyezi más programoknak, hogy használhassák azokat az értékeket, melyek ezekben a fájlokban lettek definiálva, úgy, mintha statikusan beírt TypeScript entitások lennének. Vannak harmadik féltől származó header fájlok is a népszerűbb könyvtáraknak, mint pl.: jQuery (mely az egyik legelterjedtebb JavaScript könyvtár, melynek az a feladata, hogy a fejlesztők munkáját megkönnyítse azáltal, hogy az azonos feladatokat, mint a HTML DOM manipulálása, a felhasználói eseményekre való reagálás vagy az adatok lekérése a szerverről [19]), MongoDB (egy nyílt forráskódú NoSQL adatbázis menedzselő program. A NoSQL-t a hagyományos relációs adatbázisok alternatívájaként használják [20]) és a D3.js (mely egy JavaScript könyvtár és keretrendszer, ami a vizualizációk készítésére jött létre. Az adatokat és a grafikai elemeket hozzá köti a DOM-

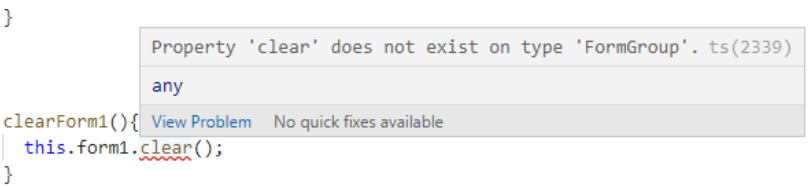
hoz és így hozza létre a vizualizációkat [21]). [22] A TypeScript header-jei a Node.js alapvető moduljaihoz szintén elérhetőek, mely megengedi a TypeScripten belüli Node.js programok fejlesztését.

Az ECMAScript specifikáció egy szabványosított specifikáció a script nyelvekhez. Hat féle kiadótt verziója létezik az ECMA-262-nek. A TypeScript az ECMAScript6-os specifikációhoz van igazítva.

8. Ábra: ECMAScript-ből TypeScript

A TypeScript adaptálja az alapvető nyelvi jellemzőit az ECMAScript5 specifikációnak, azaz, a hivatalos specifikációját a JavaScriptnek. A TypeScript nyelvi jellemzői, mint például a Modulok és az osztály alapú orientáció, összhangban vannak az ECMAScript6-nak a specifikációival. Ezenfelül, a TypeScript magába foglal olyan jellemzőket is, mint például a generikus és típus annotációk, amik nem részei az ECMAScript6 specifikációnak. [23]

Az egyik legfontosabb erénye a TypeScript-nek, amivel a fejlesztőket is meggyőzte, hogy használják, az a TypeScriptnek az összeállítása. A JavaScript egy tolmácsolt nyelv, így, szüksége van arra hogy futtatva legyen, így tesztelve azt, hogy valid, másszóval: érvényes. Ez azt jelenti, hogy ha hiba van, akkor lényegtelen a kódnak a hossza, nem fog kimenet megjelenni. Így, hibakereséssel kell a fejlesztőnek órákat is eltölteni akár. Míg, a TypeScript fordítója, egy hiba-ellenőrző funkciót biztosít, amivel a TypeScript futtatja a kódot, és legenerálja a futási hibákat, ha talál bármilyen szintax hibát. Ez segítséget nyújt abban, hogy a script futtatása előtt, már felfedezhetőek legyenek a hibák.



9. Ábra: TypeScript hiba-ellenőrző futtatás előtt

Amikor egy TypeScript szkript összeállításra kerül, van egy lehetőség arra, hogy legeneráljon egy deklarációs fájlt (.d.ts kiterjesztéssel) ami interfészként funkcionál a komponenseknek. A koncepciója az ilyen fájloknak, hasonló a C vagy C++ nyelvben megtalálható header fájlokhoz. Ezek a fájlok (d.ts kiterjesztéssel) biztosítják az intellisense-t (egy kódfeltöltés fajta, mely az elírások és más gyakori hibák csökkentésével felgyorsítja az alkalmazások kódolásának folyamatát, lehetőséget kínál az eddig beírt kódrészlet alapján a lehetséges folytatásra [24]) a típusoknak, a funkciók

meghívásának, és számos támogatást a JavaScript könyvtáraihoz, mint például a jQuery vagy a MooTools. [25]

A TypeScriptben rengeteg alapvető típusfajta megtalálható, mint a Boolean (logikai igaz/hamis), Number (számérték), String (karakterlánc), Array (tömb). Ezekből vannak olyanok is, melyek alapból a JavaScriptben nem léteznek. Ezeket kiegészítve, létezik az any (bármi) illetve az unknown (ismeretlen).

Az any lényegében minden típust képes lefedni, az unknown ennek a típusbiztos megfelelője. Amikor nem tudjuk, vagy nem akarjuk megadni egy változónak a típusát, az any megengedi számunkra, hogy bármilyen változót hozzárendeljünk. Gyakran van használva olyan esetekben, amikor egy harmadik féltől származú modell objektumról van szó, melynek a típusú még nem lett ellenőrizve, sem meghatározva. Az unknown nagyon sokban ugyanaz, mint az any, de az eltérése abban nyilvánul meg, hogy nem lehet olyan változóval semmilyen műveletet végrehajtani ami unknown-ként lett definiálva, egészen addig, amíg az nem lett típus-ellenőrizve. Érdekesség, hogy a TypeScriptben létezik egy never (soha) típus is, mely visszatérési értékként szokott előfordulni, melynek az a szerepe, hogy egy olyan esemény-t jelölünk meg vele, mely sosem fog bekövetkezni.

A metszet és az unió segítséget nyújtanak abban, hogy egyedi típust hozzunk létre. A metszet, ahogy a nevéből is következtetni lehet, lehetőséget biztosít arra, hogy több alapvető típust összevonjunk egyetlen egybe. Például, létre tudunk hozni egy Ember típust, amelynek van egy név: string, és egy kor: number tulajdonsága. Ez megegyezik azzal az állítással, hogy „ez a típus legyen ez **és** az”. Az unió lehetőséget nyújt arra, hogy a típusunk több, alapból létező típus közül legyen egy. Például, van egy lekérdezésunk, amely vagy egy string-et, vagy egy undefined (nem definiált)-at ad visszatérési értékként vissza, ez megegyzik azzal az állítással, hogy „ez a típus legyen az **vagy** az”. [26]

```
interface Bird{
    fly() : void;
    layEggs(): void;
}
interface Fish{
    swim() : void;
    layEggs(): void;
}

function getSmallPet() : Bird | Fish;

let pet = getSmallPet();
pet.
    layEggs                    (method) layEggs(): void
```

10. Ábra: TypeScriptben az unió példája egy funkció segítségével

2.7 Angular

Az Angular, mind egy platform, mind pedig egy keretrendszer, egyoldalas kliens alkalmazás fejlesztéséhez, mely HTML-t illetve TypeScript-et használ. Az egész Angular nyelv TypeScript-en íródik. Egy Angular applikáció architektúrája bizonyos alapvető koncepciókra támaszkodik. Az alapvető építőelemei az Angular keretrendszernek az

Angular komponensek, amik az NgModules-ba (NgModulok) vannak szervezve. Ezek a modulok összegyűjtenek összefüggő kódot funkcionális készletekbe, így, egy Angular applikáció NgModulok készleteként van meghatározva. Egy applikációnak mindig van legalább egy gyökér (root) modulja, ami engedélyezi a bootstrapping-et (az angular.bootstrap egy funkció komponens az ng mag modulban, melyet az Angular applikáció manuális indítására használunk, ez nagyobb hatalmat biztosít számunkra abban, hogy hogyan akarjuk inicializálni az applikációnkat [27]).

A komponensek azok, melyek definiálják a nézeteket, amik lényegében képernyő-elemek készlete, amelyek közül az Angular tud válogatni, és módísítani a program logikája és az adatok függvényében. Szolgáltatásokat használnak a komponensek, mely olyan különleges funkciókat biztosít, ami nincs közvetlen kapcsolatban a view-al. Ezek a szolgáltatás biztosítók kívülről is megadhatóak a komponenseknek, úgynevezett függőségekkel amivel a kód sokkal inkább újrahasználható és hatékony lesz. [28]

A modulok, komponensek és szolgáltatások olyan osztályok, melyek dekorátorokat használnak. Ezek a dekorátorok megjelölik a típusaikat és meta-adatot biztosítanak, melyek elárulják az Angularnak, hogy hogyan kell azokat használni. Egy komponens meta-adata társul egy sablonnal, amely definiálja a nézetet. Egy sablon egyesíti a hagyományos HTML-t az Angular irányelveivel és kötelező jelölésével, mely lehetővé teszi az Angular számára, hogy átalakítsa a HTML-t mielőtt azt megjelenítésre felmutassa. Egy szolgáltatás meta-adata biztosítja a szükséges információt az Angularnak ahhoz, hogy az képes legyen elérhetővé tenni a komponensek számára függőségi injekció segítségével.

Egy alkalmazás komponensei tipikusan több nézetet határoznak meg, melyek hierarchikusan vannak elrendezve. Az Angular egy router szolgáltatást biztosít, melynek segítségével tudjuk definiálni a navigációt, útvonalak a nézetek között. A router kifinomult, böngészőben levő navigációs képességeket biztosít.

Az Angularban megtalálható NgModulok különböznek és kiegészítik a JavaScriptben megtalálható modulokat. Egy NgModul kijelent egy gyűjtemény kontextust egy adott komponens készlethez, amely egy alkalmazás domainnek, egy munkafolyamatnak vagy egy szorosan összefüggő képesség-készletnek lett dedikálva. Egy NgModul képes összetársítani a komponenseit hasonló kóddal, mint például egy szolgáltatással, azért, hogy funkcionális egységeket hozzon létre.

Minden Angular applikáció rendelkezik egy root modullal, mely hagyományosan AppModule-nak van elnevezve a programon belül. Ez a modul biztosítja a bootstrap-et, amellyel elindítjuk az applikációt. Tipikusan számos modul megtalálható egy Angular applikációban. Ahogyan a JavaScriptben található modulok esetében, úgy a NgModulok is tudnak importálni funkciókat más NgModultól, valamint engedélyezni azon saját funkcióinak, hogy exportálják és felhasználják más NgModulok is. Erre a

legkézenfekvőbb példa, hogy ha használni akarjuk a router szolgáltatást, akkor egyből importáljunk kell a Router nevű NgModult.

Ha úgy rendezzük a kódunkat, hogy különböző funkciók szerint külön modulokba kiszervezzük az egyes részeket, sokkal átláthatóbb, és újrafelhasználhatóbb kódot kapunk. Továbbá, ez a technika lehetőséget biztosít arra, hogy kihasználjuk a „lazy loading”-ot (lusta betöltés), amelyben csak az igénybe vett modulokat töltjük be, ezzel is minimalizálva azt a kódmennyiséget, melyet futtatni kell az applikáció elindításakor.

Minden Angular applikációban megtalálható legalább egy, a router komponens, mely összekapcsolja a komponens hierarchiáját az oldal dokumentum objektum modelljével. Minden egyes komponens definiál egy osztályt, ami tartalmaz applikációhoz tartozó adatot, logikát és össze van társítva egy HTML sablonnal, ami előírja a nézetet, amit meg kell jeleníteni a célkörnyezetben. A ***„@Component()”*** megjelölés fogja azonosítani az osztályt rögtön alatta, hogy az egy komponens és biztosítja a sablont valamint a kapcsolódó komponens specifikus meta-adatot. Ezek a megjelölések olyan funkciók, melyek módosítják a JavaScript osztályokat. Az Angular számos ilyet definiál, amik meghatározott fajtájú meta-adatot csatolnak hozzá osztályokhoz, azért, hogy a rendszer tudja, hogy mik azok az osztályok, valamint, hogy hogyan kell nekik működni.

Az Angular dekleratív egyirányú adatkötést biztosít, mint alapértelmezett viselkedés (ezt persze ha kívánjuk, át lehet állítani két-irányú adatkötésre). Az egy-irányú adatkötés egyirányú változásterjedésként viselkedik amely teljesítménybeli javulást, valamint a kód komplexitásának csökkentését biztosítja. Továbbá, az Angular támogatja az állapottartó, reaktív, valamint megváltoztathatatlan modelleket is.

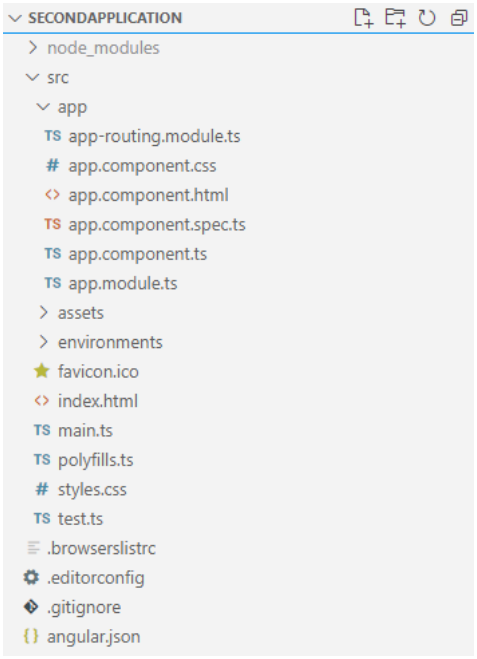
Az Angular applikációk magukba foglalják a legfelső szinten levő „***root***” modul definiálását, ami egy komponensre hivatkozik, ami pedig meghatároz egy HTML elemet, ami a szülő eleme a komponensnek. A sablon tulajdonság tartalmaz egy keveréket a HTML fájlból, valamint egy egyedi megjelölést, amit egy külön fájlban lehet elhelyezni, és aztán arra a fájlra hivatkozni a „***templateUrl***” tulajdonsággal. Továbbá, a komponenst azonnal követi egy TypeScript osztály definíció, mely tartalmazza a háttérkódot, amely társult azon komponenssel-kapcsolatban álló változókkal, amik megjellenek a sablon tulajdonságban. [29]

Az Angular applikációk beépített és egyedileg - amit a fejlesztő magának alkotott meg - elkészített komponensek kombinációjából áll, melyek mindegyike tipikusan egy külön TypeScript fájlban (.ts kiterjesztéssel) van definiálva. Minden egyes komponens egy vagy több import állítást használ, hogy tartalmazni tudják a függőségeiket.

Számos variáció típus létezik a függőségekre az Angular-on belül, mint például: directive (irányelv) és pipes (csövek). Egy egyedi irányelv alapvetően a tartalma egy TypeScript fájlnak, ami definiálja a komponenst. Így, egy egyedi irányelv, import állításokból, egy komponens dekorátorból és egy exportált TypeScript osztályból áll.

Az Angular beépített irányelveket is biztosít, mint az „***ngIf***” („ha” feltét) és az „***ngFor***” (ciklusokhoz). Ezt a két irányelvet szokás még strukturális irányelveknek nevezni, mert módosítják a HTML oldal tartalmát. Az Angular beépített pipe-jai tartalmaznak dátum és szám – pl.: decimális, szám- formátumokat, míg az egyedieket a fejlesztő készíti el.

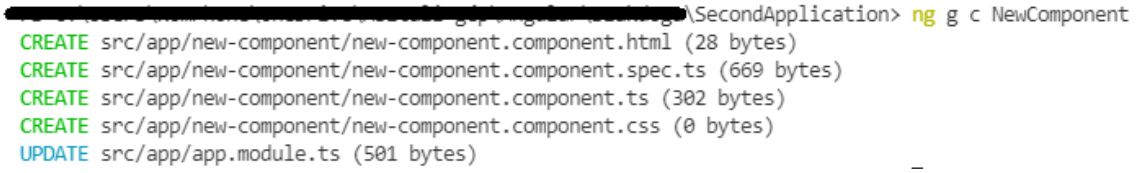
Továbbá, a TypeScript osztályok dekorátort használnak (ami egy beépített funkció), ami meta-adatot biztosít az osztálynak, annak tagjainak, vagy a metódus argumentumainak. A dekorátorokat könnyen lehet azonosítani, mert mindig tartalmaznak egy „@” előtagot. A leggyakrabban használt dekorátorok a „***@Component***” és az „***@NgModule***”.



11. Ábra: Angular applikáció elkészítése után a kezdeti struktúrája a fájloknak

Amikor egy új Angular projektet hozunk létre számos fájl és lehetőség fogad minket, amikor először megnyitjuk (jelen esetben a Visual Studio Code-ot használva). Ahogyan már korábban részletezésre került, három főbb fájlja van, egy-egy komponensnek. Egy .html kiterjesztéssel rendelkező fájl, mely az adatok és funkciók megjelenítéséért felel. Valamint egy .css ami stílusilag alakítja a HTML elemeket, nagyobb irányítással, mint ha csak a HTML-t használnánk erre. Négy évvel a HTML után készült el, teljesen elszeparáltan fejlesztették ki a HTML-től, így meghagyva a lehetőséget a fejlesztőknek, hogy csak HTML-t használva is tudjanak oldalakat létrehozni. Alapvetően, néhány CSS funkció átfedett már létező HTML funkciókat, azonban a stílushatások megadása a HTML kódban zsúfolt és rendezetlen kódot eredményezett. [30]

Korábban már esett szó a „.ts” kiterjesztéssel rendelkező fájlokról, mint a TypeScript nyelvű fájlok. Minden érdemi fejlesztés ezekben a fájlokban történik. Amikor létrehozunk például egy új komponenst, akkor alap esetben létrejön négy fájl: létrejön a



12. Ábra: Angular applikációban egy új komponens létrehozása

már tárgyalt .html kiterjesztésű, a .css, valamint a .ts fájl is. Ezeken kívül, ha a fejlesztő mást nem tesz, létrejön egy .spec.ts fájl is, melynek a szerepe a komponens tesztelése lesz. Amikor nem akarjuk, hogy létrejöjjön a tesztelésre szánt fájl, arra a „***—skip-tests=true***” kifejezést tudjuk használni a terminálon belül, amikor létrehozzuk. Ilyenkor a keretrendszer automatikusan kihagyja annak a fájlnak a legenerálását. A parancsban szereplő „c” a „component”-et jelenti, nem pedig az osztályt.

Az Angularban számos életciklus metódus megtalálható, melyek előre meghatározott sorrendben hajtódnak végre az applikáció élete során. A két leggyakrabban használt, és egymással szorosan összefüggő ilyen metódus az „***OnInit***” és az „***OnChanges***”, pontosabban, az „***OnInit***” és az „***OnChanges***” interfészek, melyeknek egy metódusa létezik, aminek a neve az interfész nevével megegyezik, csak az elején egy ng-vel kiegészítve, így ezekből az utóbbi akkor hívodik meg, amikor egy adatkötött bemeneti vagy kimeneti adat megváltozik, előbbi pedig pontosan azután hívodik meg, amikor az első „***OnChanges***” lefut.

```
export class NewComponentComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```

13. Ábra: ngOnInit megjelenése egy kódban

Az „***OnInit***” interfészt a komponens magától azonnal implementálja, viszont amikor egy modellt reprezentáló osztályt hozunk létre, (függetlenül attól, hogy van-e hozzá tesztelésre szánt fájl) akkor ott nem implementálódik az interfész.

A Socket.IO egy olyan JavaScript könyvtár, mely engedélyezi a valós idejű, kéitrányú és esemény alapú kommunikációt a böngésző és a szerver között. [31] Ahhoz, hogy létrehozza a kapcsolatot és hogy adatokat tudjon cserélni a szerver és a kliens között a Socket.IO az Engine.IO-t használja, ami biztosítja az alacsonyabb szintű kapcsolatot a szerver és a kliens között. Az Engine.IO-t használjuk a szerver implementálásához és az Engine.IO-client-et a klienshez. A Socket.IO hasonlítható a WebSocket-ekhez. A

WebSocket egy kommunikációs protokoll ami egy teljes-kétirányú és alacsony késleltetésű csatornát biztosít a szerver és a böngésző között, de a Socket.IO nem ezt használja alapértelmezettként. [32]

A Socket.IO lesz képes arra, hogy egyszerre több klienst szimultán ki tudjon szolgálni, valamint amikor megváltozik az oldal, vagy a kliens egy eseményen keresztül interaktol a szerverrel és ez választ generált, akkor a felhasználónak nem kell frissíteni az oldalt. Számos olyan funkciót biztosít a Socket.IO az Engine.IO-n keresztül, melyet az alap WebSocket-ek nem tudnak, ilyen például: automatikus újracsatlakozás, közvetítés minden jelen levő felhasználónak, vagy csak egy adott részhalmazának a felhasználók közül, amit szobának nevezünk.

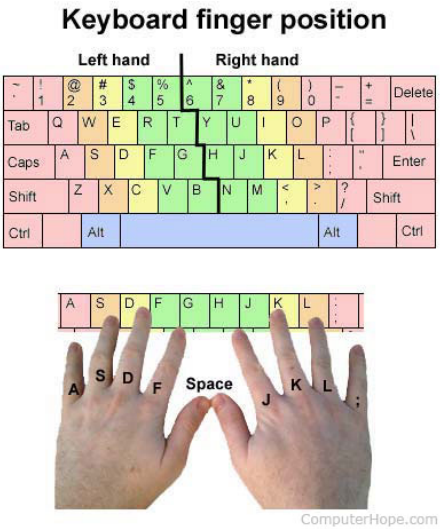
2.8 Kiértékelő logika

Minél hosszabb, azaz több betűből áll egy szó, annál nehezebb leírni azt. Legyen szó tollal papírra való írásról, vagy billentyűzettel gépelésről.

Egy átlagos angol szó hossza öt betűből áll egy 2012-ben készült tanulmány szerint. [33] A tanulmányban az is fellelhető, hogy fiktív szavak kevésbé ismertek. Egy olyan szót, melyet az adott egyén már ismer betűről betűre, legyen az anyanyelvével megegyező nyelven levő szó, vagy idegennyelvű, azt az ember könnyebben le tudja írni, hisz nem telik el idő két betű leírása között. Így a szakdolgozatomban ez is súllyal rendelkezik, meg kell határozni, hogy egy szót mekkora eséllyel ismerhet alapjáraton is a felhasználó.

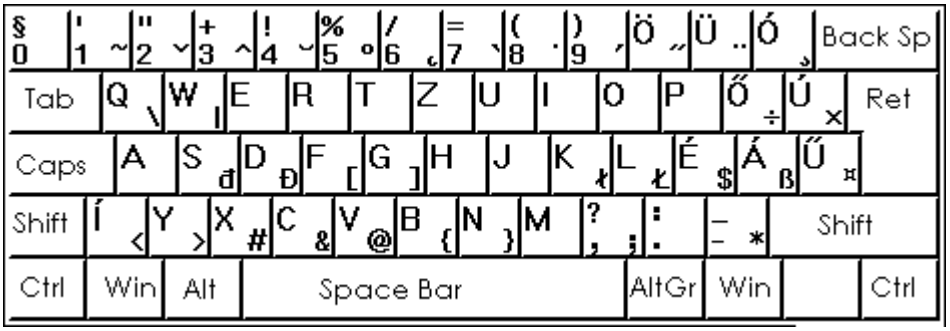
Az Oxford egyetem évekkel ezelőtt közzé tett két listát, melyben a leggyakrabban használt háromezer, illetve ötezer angol szót jelölték meg, ezzel is segítve az angol nyelvet elsajátítókat. Az alapján választották ki ezeket a szavakat, hogy milyen gyakran fordultak elő az Oxford English Corpus-ban, mely egy szótárnak felel meg, valamint, hogy mennyire relevánsak ezek a szavak a mindennapi életben. A legtöbbet használt háromezer szavas verzió nyelvvizsga tekintetében A1-B2-es kategóriának felel meg, az ötezer szavas kiadás pedig már a C1-es kategóriába tartozik inkább bele. [34]

A harmadik befolyásoló tényező pedig az egyes szavakban megtalálható betűk. Minél nagyobb fizikai távolság van a billentyűzeten az egymás után következő szavak között, annál tovább tart egy szót leírni. Vegyük példának az angol „nod” (jelentése: bólintás). Egy billentyűzeten, a szó mindhárom karaktere meglehetősen messze van egymástól. Itt jön be viszont, hogy a mai, huszonegyedik században az átlagos felhasználó nem egy kézzel gépel, hanem kifejlesztette azt a technikát, hogy mindkét kezével használja szimultán a billentyűzet gombjait. Egyénfüggő az, hogy az ember hogyan helyezi el a kezeit a billentyűzet felett, de jó esetben kellő távolság van köztük, így minél nagyobb területet képesek lefedni a gombokból.



14. Ábra: Javasolt kézelhelyezés gépeléshez

Ha vesszük a 14. ábrán javasolt kézelhelyezést, akkor az angol abc 26 betűjét két részre osztjuk, 15-11 arányban. Ehhez persze hozzá tartoznak a nem betű karakterek, de a szakdolgozatomban arra nem szeretnék kitérni. Közös billentyű azonban a szóköz, melyet körbekérdezési tapasztalat alapján az emberek nem szenteltek dedikáltan egy ujjnak, hanem helyzettől függőnek határozták meg.



15. Ábra: Magyar nyelvű billentyűzet kiosztása

Ha megfelőlen, azaz az ajánlott elhelyezéssel használjuk a kezünket, akkor nyolc fontosabb dedikált gombunk lesz, mely gombért, és a körülötte levőkért egy ujj fog felelni. Az „A” körül levő foglalja magába önmagát, az „A”-t, valamint a szakdolgozatomban mérvadó billentyűk közül még a „Q;Í;Y”-t. Az „S” esetében rajta kívül a „W;X” páros esik az adott környezetbe. Ahogy haladunk a billentyűzet közepe fele, úgy kap egy-egy környezet több billentyűt amiért felelős. Átlagosan így egy környezetbe, amiért egy dedikált ujj felel, 3.25 billentyű jut. [35]

Ezek által meghatározásra kerültek az egyes környezetek, amikből számítható ki a távolság. Mivel nem egy kézzel gépel az ideális felhasználó, ezért a távolság, mint szó, elhanyagolható. Innentől az fog számítani, hogy az egymás után következő betűk egy környezetbe esnek-e. Ha igen, akkor azt nem tudja a felhasználó párhuzamosan, gyorsan

leírni, mert minden ilyen betűt egyesével tud csak begépelni. Azonban, ha a betűk más és más környezetbe esnek bele, akkor szinkronban lehet folyamatosan gépelni, meggyorsítva ezzel jelentősen a folyamatot.

Ez a három főbb tényező lesz, mely lényegében használat után az eddigi teljesítményt méri, valamint így minden szóhoz megállapítható lesz egy nehézségi szint. Ezen nehézségi szint meghatározásához mindhárom tényezőt figyelembe kell venni, mindegyiket súlyozottan.

Az első kategória amit súlyozni kell, az a szó hosszúsága. Minél több betűből áll egy szó, annál nehezebb. Mivel az átlag szóhossz az angol nyelvben öt betűből áll, ezért az lesz az "aranyközépút", így tehát, ha egy szó 5-7 betűből áll, annak a súlya 0.3 lesz. Ha valamelyik szó több mint 7 betű hosszúságú, akkor annak a súlya 0.4, ellenkező esetben viszont, ha az átlagosnál kevesebb betűből áll egy szó, akkor ezen tekintetben levő súlyozottsága mindösszesen 0.2 értékkel bír a képletben. Abban az esetben viszont ha egy nagyon hosszú szó van soron, 0.55-ös súlyozást fog kapni.

A második szempont a szónak az ismertsége volt. Négy lista áll rendelkezésre, ebből kettőt az Oxfordon határoztak meg. Ezek 500, 1000, 3000, illetve 5000 szavas listák, melyből minél kisebb számú listában szerepel egy szó, annál inkább ismertebb, ennélfogva, annál könnyeben lehet leírni. Ebből adódóan, ha egy szó annyira ismert és használatos, hogy az 500-as listában helyet kapott, akkor a súlyozott értéke ebben a szempontban 0.1 lesz. 1000-es listába való kerülés esetén 0.2. Ha egy vizsgált szó a 3000-es listában található meg, akkor az előző kategóriához mérten a 0.4-es értéket kapja meg. Abban az esetben viszont, ha egy szó csak az 5000 szavas listában került említésre, akkor azt jelenti hogy nagyon ritkán használt szó, a többihez képest, ebben az esetben a súlyozása 0.5 lesz.

Az utolsó szempont melyet figyelembe kell venni az az adott szóban levő környezetek száma. De az nem elég, hogy hány környezet van, ahogy az sem, hogy hány betűből áll egy szó. A legfontosabb megvizsgálandó pont ezen a szemponton belül, hogy az egymás után következő betűk azonos környezetbe esnek-e. Ha igen, akkor azt nem számítjuk bele többszörösen. Így ahol például hosszú mássalhangzó, avagy dupla magánhangzó szerepel, máris nagyobb értéket fog kapni a végső nehézségi mutatóban. Minden szónál meg kell a jelenlegi szemponthoz vizsgálni azt, hogy tehát hány karakterből áll, hány környezetet érintenek ezek a betűk, ezt a két összeget elosztjuk egymással és a normalizálás végett beszorozzuk egy 0.15-ös állandóval.

2.9 Összegzés

A szakirodalomban levő kutatásom alatt megvizsgáltam, hogy az internet milyen protokollok alapján működik, valamint milyen lépések mentén és hogyan jut el egy átlag felhasználó a böngészőjén keresztül egy webalkalmazásra.

Hasonló rendszereket kutattam fel, melyek lényegi részben eltérnek, de mégis ugyanazon kategóriába tehetőek mint a szakdolgozatom témája. Ebből részletezésre került két kategória: az emélekezettel rendelkező, valamint anélküli rendszerek. Mivel a szakdolgozatom témája az előbbivel rendelkezik. Mindegyik példának megismertettem a hátrányát, valamint egyes előnyeit a témához kapcsolódóan.

Backend részre kutattam az általam kiválasztott fejlesztői környezeten kívüli lehetőségeket, de a széles körben elterjedt ASP.NET-re esett a választásom, végsősoron azért is, mert egyetemi tanulmányaim során volt már ilyen tárgyam. Számos remek lehetőséget kival fel az ASP.NET, valamint egy általam már ismert, C# nyelven lehet dolgozni benne.

Frontend téren megvizsgáltam a JavaScript-et, mert ez a nyelv a legszélesebb körben elterjedt programozási nyelv webfejlesztés területén. Számos erre épülő technológia van, valamint sok forrás is elérhető hozzá. Az egyik erre épülő a TypeScript, amire az Angular, mely az egyik, ha nem a legnépszerűbb fontend fejlesztői keretrendszer napjainkban. Emellett szerverek területen megvizsgálásra került a Node.js, melyben fejlesztői oldalon igaz, hogy nem szükséges a JavaScript tudáson kívül egyéb.

Mivel a szakdolgozatom témája egy adaptív pontozási rendszert alkalmaz a szavak nehézségének meghatározására, ehhez utána kutattam, mely szavak a leggyakoribbak az angol nyelvben, ez volt az egyik súlyozási kritérium, valamint a szavak hosszúsága, és egy egyéni elgondolás amivel a végső képlet meghatározásra került. Ezek fogják alkotni a nehézségét az egyes szavaknak.

3. Követelmény specifikáció

A rendszernek képesnek kell lennie szimultál kiszolgálni több felhasználót is. Ezek közül ha két adott felhasználó azt kívánja, biztosítania kell egy úgynevezett szobát, melybe csak ők ketten tudnak belépni, és az oldalt a továbbiakban úgy tudják használni, hogy egymás ellen mérik fel tudásukat. A rendszernek alapvetően rendelkeznie kell egy adaptív pontozási rendszerrel a felhasználók fele, mely alapján történnek a használat közben megjelenő szavak és feladatok.

A felhasználóknak kötelességük regisztrálniuk az oldalra ahhoz, hogy használni tudják. Ezt a rendszernek egy adatbázisban tárolnia kell, valamint az adott felhasználóhoz eltárolni az eddigi összes – ha nincs, létrehozni ennek egy megfelelő adatszerkezettel rendelkező tárolást – eddigi eredményét, melyet a felhasználó el is tud érni az egyik menüpontjában.

A rendszernek ezen túl, a felhasználó fele lehetőséget kell kínálnia egy részletesebb eredmény közlésre, melyben az aktuálisan megjelent szavak nehézsége, valamint a felhasználó végső elemzése is megtalálható, melyben részletesebben láthatja a felhasználó az adaptív pontozási rendszer működésének elvét.

A rendszernek azon túl, hogy egy privát szobát létre kell tudnia hozni, egyéni használatra is lehetőséget kell hogy biztosítson egy felhasználónak abban az esetben, ha gyakorolni szeretne, vagy csak egyedül szeretné használni az oldalt, azaz ne legyen kötve az oldal valós használata másik felhasználóhoz.

A rendszer fele további elvárás, hogy a felhasználó szabadon megszakíthassa az adott, már megkezdett felhasználást, ilyenkor az abban az időszakban kapott szavak, és minden eredmény törlésre kerüljön, és ne számítódjon bele a felhasználó eredményeibe, valamint ne befolyásolja egy másik szobába való belépését, vagy az egyedüli használat elkezdését.

4. Tervezés

4.1 Rendszerterv

A rendszer specifikációinak megfelelve, a rendszer három főbb modulra bontható melyek a kliens oldal, a szerver oldal, illetve egy adatbázis. Az első kettő, azaz a kliens illetve szerver oldal közötti kapcsolatnál az elsődleges kommunikáció a REST API segítségével, valamint a tervek szerint JSON fájlokkal történik meg. A szerveroldal, valamint az adatbázis modulok között az egyetemi tanulmányok alatt megismert Entity Framework fogja a kapcsolatot létrehozni.

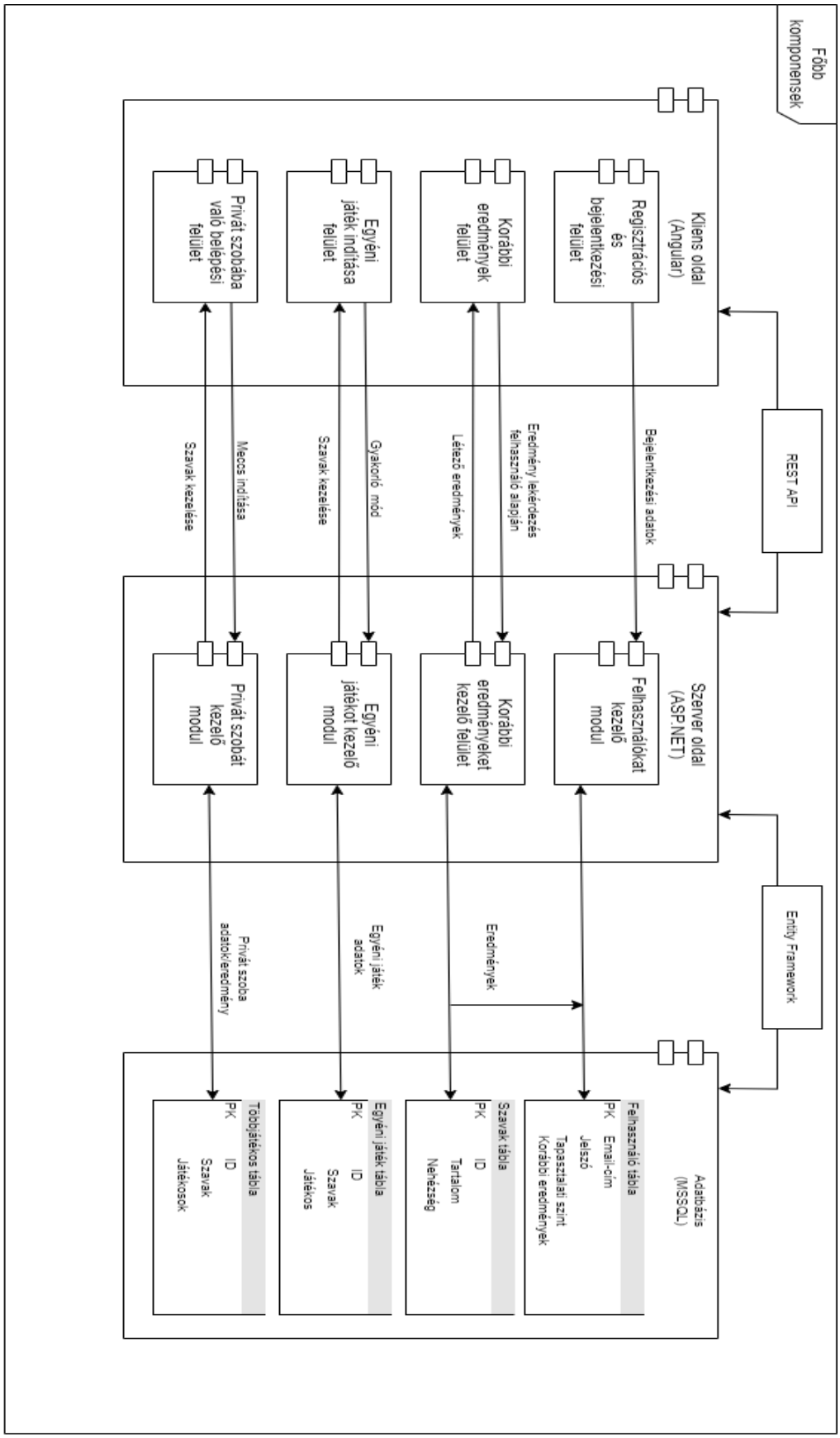
Ahogyan a 16-os ábrán is látható, mind a felhasználók, mind a szavak külön táblát kapnak az adatbázisban. Ezt persze lehetne mellőzni, és a szavakat csak hozzárendelni a felhasználóhoz, viszont ebben az esetben az adaptív pontozási rendszert megvalósítani a későbbiekben szignifikánsabban nehezebb lenne, valamint az alapvető OOP[17] elvet sem tartaná be ez a megvalósítási forma.

A felhasználó a sikeres bejelentkezés, vagy regisztráció után – mely ellenőrzésre kerül az adatbázison belül – rendelkezik a lehetőséggel ahhoz, hogy az eredményeit megtekintse, vagy hogy a két játékmód közül választhasson, és azt elindíthassa.

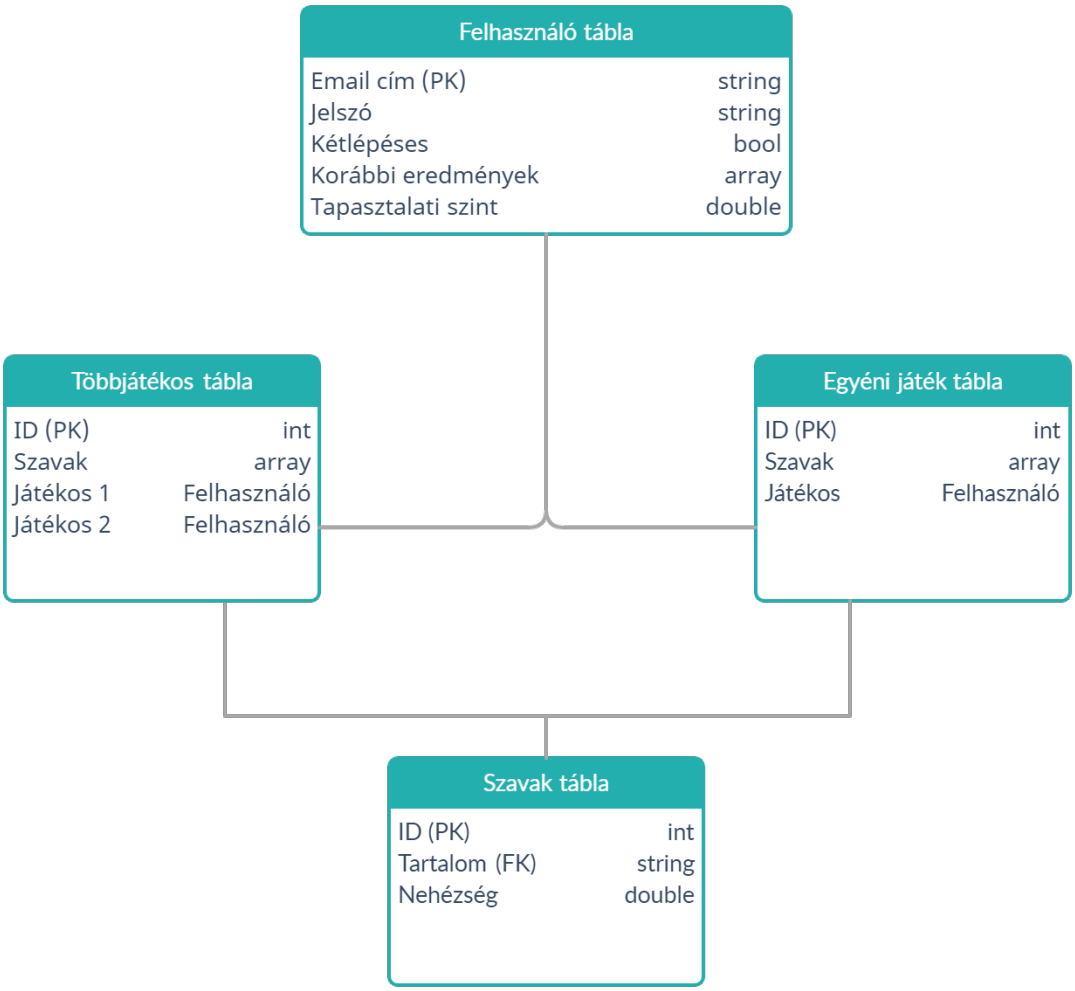
Ezen felhasználó kezelések, eredmények lekérése, összegzések, valamint az új adatok adatbázisba juttatása egyszerű CRUD[18] műveletekkel történnek.

[17] Objektum Orientált Programozás, egy programozási paradigma, amely a program egészét objektumok halmazából építi fel

[18] Create-Read-Update-Delete, azaz létrehozás, lekérdezés, módosítás, törlés, alapvető műveletek



16. Ábra: Főbb komponensei a rendszernek



17. ábra: A rendszer adatbázis modellje a tervezés fázisban

A 17-es adatbázist reprezentáló ábrán jól látható, hogy például a Szavak táblában a végső nehézséget csak egy mezőben tárolja el a rendszer. A rendszer tervezésénel nem szerettem volna azt, hogy minden, a nehézséget befolyásoló tényező külön tulajdonságot kapjon az osztályon belül, hanem amikor beolvasásra kerül, vagyis az adatbázisban hozzadásra kerül egy szó, akkor hajtódjon végre rajta a korábban már részletezésre került adaptív képlet, és egyszerű, lebegőpontos számként jelenjen meg a végső érték.

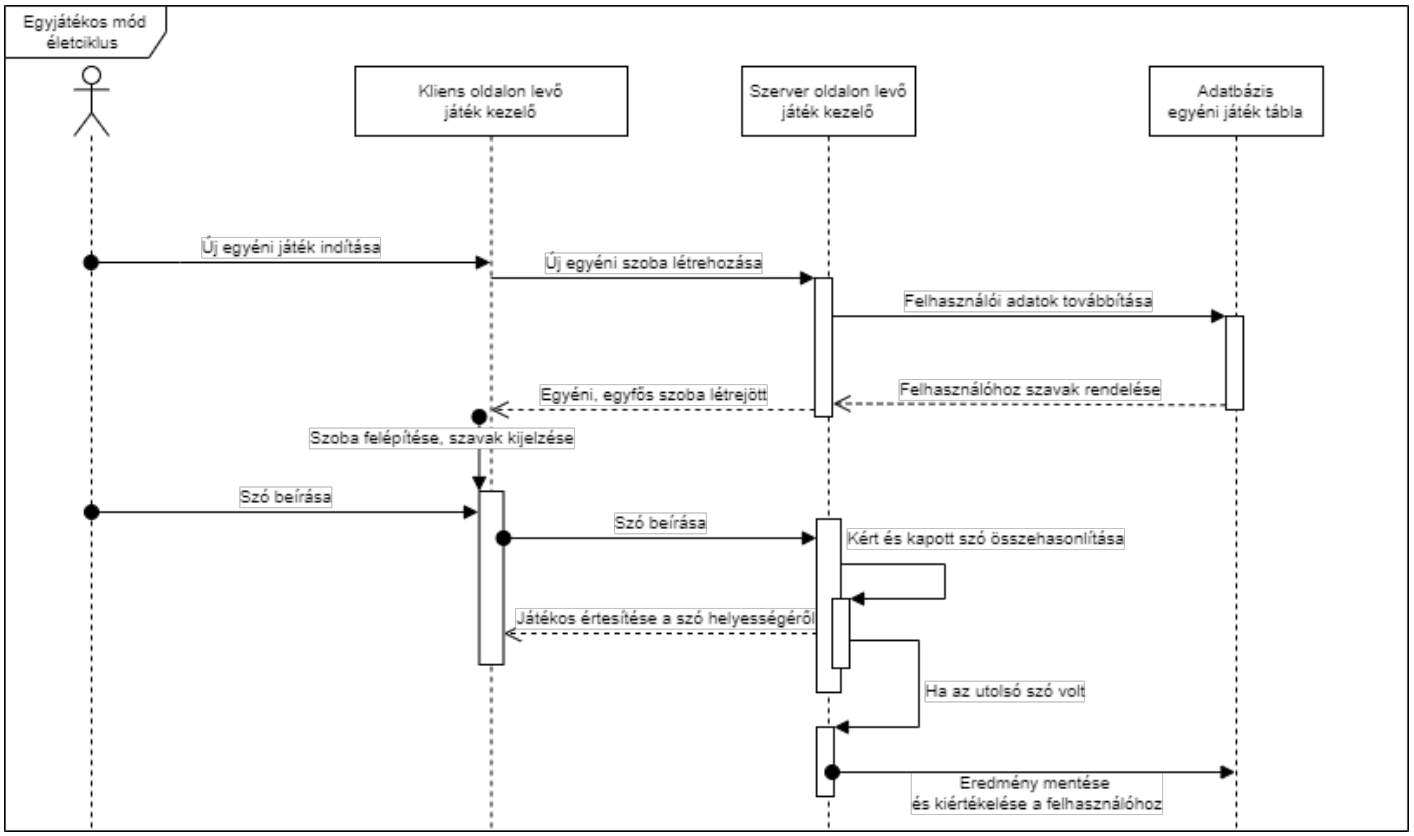
Az adatbázis modell középpontjában a Szavak tábla áll, mely mindegyik módhoz kapcsolva van, ahogyan a felhasználó is.

4.2 Funkciólista

A rendszernek az alábbi funkciókkal kell rendelkeznie:

- A regisztráció valós email cím használatával történjen
    - valós email cím alatt olyan email címet tud elfogadni a rendszer, amely a szabványos és ismert @xyz.com/hu végződéssel rendelkezik, egyéb esetben nem jó formátumra hivatkozzon a rendszer és a felhasználót értesítse erről
- A bejelentkezésnél legyen lehetőség kétlépéses bejelentkezésre
    - itt a valós sms-t küldő, vagy telefonos applikációban megjelenő kód-tól eltérően a felhasználónak lehetősége legyen egy olyan funkciót használni, ami bejelentkezésnél nem csak a jelszavát kéri el az email cím mellett, hanem egy egyedi kérdésre is válaszolnia kell
- A felhasználónak legyen lehetősége az adatait változtatni bejelentkezés után
- A felhasználók tudják a rendszeren lekérni a korábbi eredményeiket, és azok részletesebb kiértékeléseit
    - részletesebb eredmény alatt itt a rendszernek biztosítania kell az adott folyamatban kapott szavak felmutatását, a felhasznált időt, és hogy mikor történt, utóbbit csak a felhasználó nyugtázása végett
- Legyen lehetősége a felhasználónak a manapság elterjedt Dark Mode[19] használatára, valamint arról való visszaváltásra az általánosra
- A felhasználónak legyen lehetősége egyéni játék módot indítani
    - ilyenkor a rendszer nem dobhat rá hibát, hanem egy egyszemélyes szobát hoz létre a szerver a felhasználónak, melyet lehetősége van megállítani is
- Többjátékos módot is tudjon a felhasználó indítani
- Már létező szobába legyen lehetősége a felhasználónak belépni egy egyedi kód megadásával
    - amennyiben ebben az esetben a felhasználó hibás kódot ír be, a szobába ne tudjon becsatlakozni mindaddig, amíg megfelelő kódot be nem ír, erről egy visszajelzést küldjön a felhasználónak a rendszer, és léptesse vissza a főoldalra
- A tervezés szerint legyen lehetősége a felhasználónak új, egyéni szavakat bevinni a rendszer adatbázisába
    - ez a funkció kezdetleges ötlet, de a megvalósítás fázisban szeretném ha helyet kapna, ehhez is az adaptív rendszert alkalmazva

[19] Sötét mód, a böngészőben megjelenő oldal átalakítása úgy, hogy a legkisebb fénymennyiséggel működjön

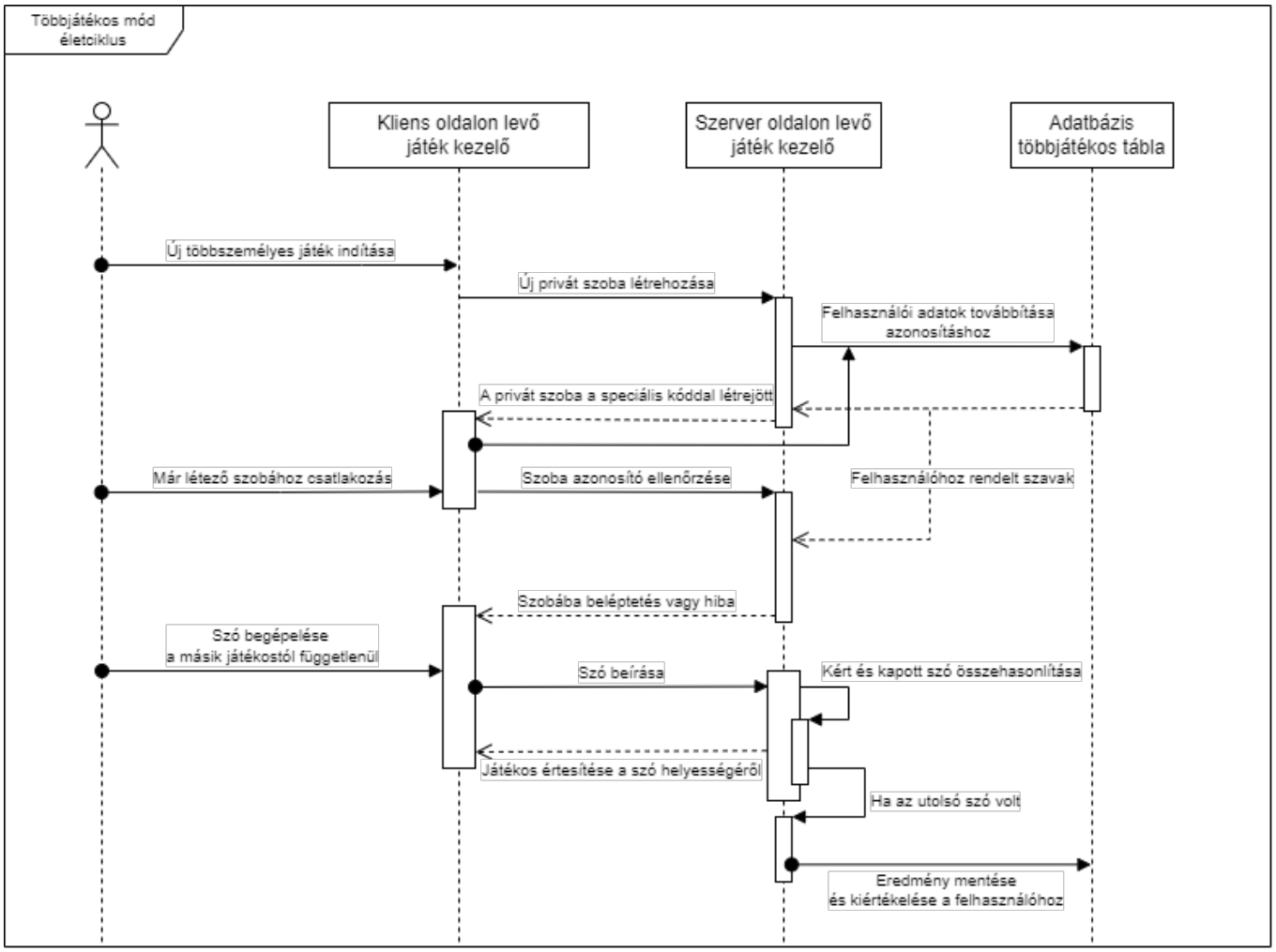


18. Ábra: Egyszemélyes mód folyamata

Az egyszemélyes játékos mód interakcióit és lehetőségeit a 18-as ábrán szemléltettem. A felhasználónak az egyéni módban, ahol nincsen másik játékos vele egy dedikált szobában, lehetősége van először is létrehozni ezt a módot, ahol egy egyszemélyes szoba kérést intéz a szerver fele. Ezen kérés feldolgozása több fázisból épül fel: a felhasználót meg kell vizsgálni aszerint, hogy eddig milyen eredményeket ért el, milyen nehézségi szintet kell belőnie neki a szavak területén a rendszernek, és ezen szavakat hozzá kell rendelnie a szobához és a felhasználóhoz.

A felhasználónak ilyenkor elkezdődik a folyamat, melyben a szavakat fogja megkapni. Ekkortól, a rendszernek végig biztosítania kell egy megállás opciót, amikor az időszámítódás is abbamarad, így ha a felhasználó egy perces szünetet szeretne tartani, akkor az az időkiesés nem lesz hatással az adaptív pontozásra.

Amikor az adott szobához rendelt idő elfogy, vagy a felhasználó az összes szót beírta, a szoba megszűnik, ezt a szerver feldolgozza, és a felhasználó az eredményeiről kap egy értesítést, egy visszajelzést afelől, hogy hogyan teljesített, amit az adatbázisba a rendszer el is ment.



19. Ábra: Többszemélyes mód folyamata

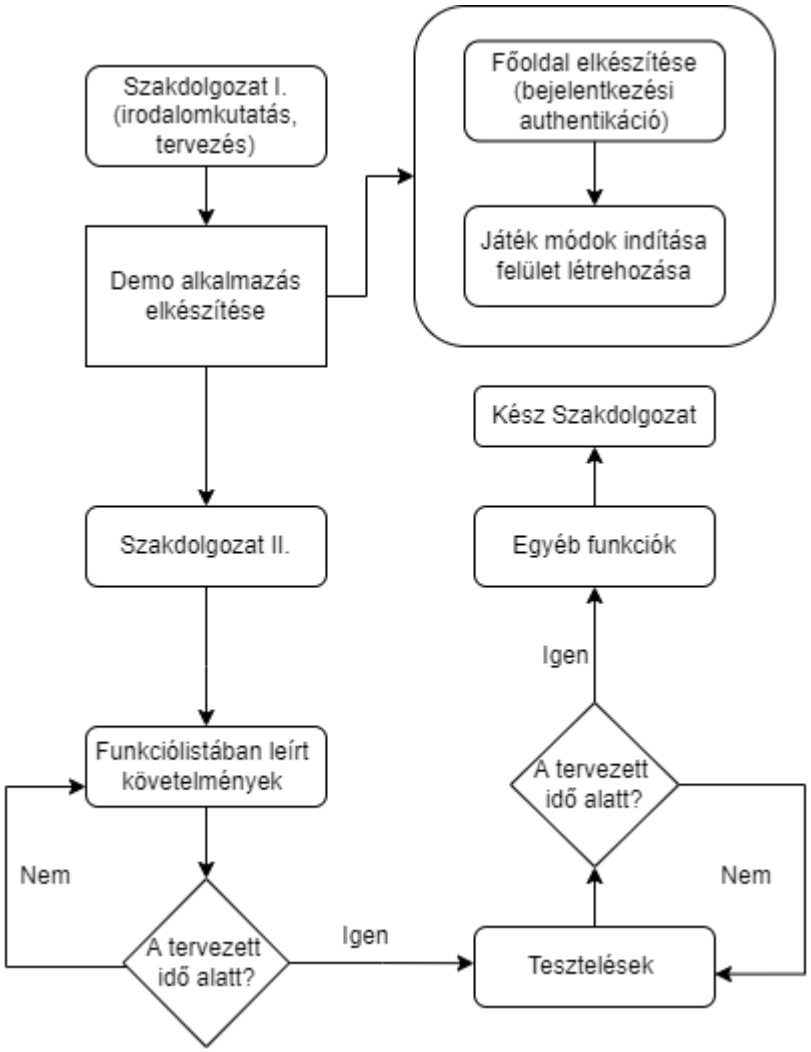
A 19-es ábrán látható többszemélyes mód valójában nem sokban tér el az előbb részletezett egyszemélyes módtól. Mindazonáltal fontos megemlíteni, hogy ilyenkor a felhasználónak két lehetősége is van: létrehozni egy privát szobát, vagy csatlakozni egyhez. Amennyiben az utóbbit választja, a szobának a kódját a szerver oldal ellenőrzi le, és az ellőrzés után a felhasználót vagy belépteti a szobába, ahol egy rövid visszaszámlálás után elkezdődik a játékuk, vagy pedig egy hibával tájékoztatja a felhasználót, hogy az imént beírt kód hibás.

A rendszer amikor többjátékos mód jön létre, mindkét felhasználót azonosítja, és mindkettejükhöz hozzárendeli egyedileg a szavakat, vagy akár ugyan azon szavakat is kaphatják, csak más sorrendben, nehézségtől és szinttől függően.

Ha az egyik felhasználó végzett, őt azonnal tájékoztatja a rendszer az eredményről amiket az adatbázis el is tárol, viszont a másik, még nem végzett felhasználóra ez nincs hatással. Mindazonáltal, felmerülhet az is, hogy egy felhasználó elhagyja fizikailag a számítógépet, ilyenkor a rendszer egy előre definiált időlimitet vár, majd a felhasználót kilépteti mind a szobából, mind a felhasználói fiókjából.

4.3 Fejlesztés tervezett menete

Irodalomkutatásom befejezésével egyből a fejlesztés fázisba tervezek lépni, és az alapoktól felépíteni a rendszert. Hiszen egy Angular frontend és ASP.NET backend összekapcsolása során számos probléma felmerülhet, melyekre jobb idő előtt fényt deríteni. Terveim szerint a Szakdolgozat dokumentációban felállított rendszer-követelményeket a félév első hét-kilenc hetében sikerül teljesítenem fejlesztés szempontjából, miután a tesztelések fázis következne. Ezek után, amennyiben sikeresen tudok haladni a fejlesztéssel és az időbeosztás tartásával, egyéb, jelenleg még nem kitalált felhasználói élményt növelő funkciók kerülnének esetlegesen beiktatásra.



20. Ábra: A fejlesztés tervezett menete

5. Fejlesztés

5.1 Szerver

Az alkalmazás szerver oldalát, azaz backend részt az irodalomkutatás során már vizsgált ASP.NET MVC formájában készítettem el. Ez API végpontok segítségével képes kommunikálni a webes felülettel. A rétegzési elvet betartva, három fontos réteget különböztettem meg, melyek a Model, a Logic és a Controller. A Model-ben helyezkednek mind az egyedosztályok melyek az adatbázis létrehozásához is kellettek, mind pedig a DTO[20]-k is. A Logic réteg felel az üzleti logikáért, és köti össze a végpontokat az adatbázissal, valamint az osztályegyedekkel. Az egyes számítási és logikai metódusok is ebben helyezkednek el. A Controllerben helyezkednek el az API végpontok, melyeket a webes felületen keresztül ér a felhasználó és kommunikál a szerverrel.

5.2 Adatbázis és modellek

Ahogy már említésre került, a Model rétegben helyezkednek el az egyedosztályok, és majd az adatbázist képző minden osztály. A végső adatbázisban az alábbi táblákat és adatokat helyeztem el az alkalmazás számára:

Words: A szavak listája, melyben minden rekord rendelkezik egy egyedi azonosítóval, a saját jelentésével és nehézségi szintjével.

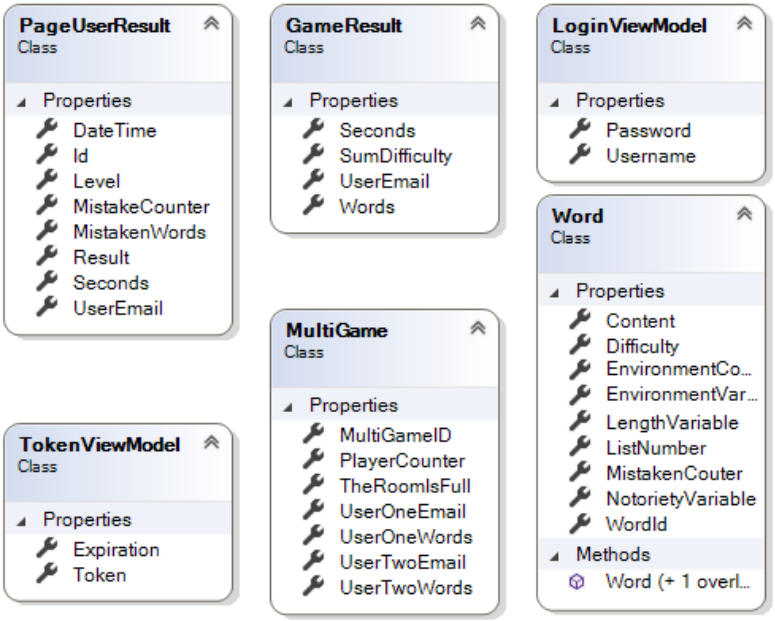
PageUsers: A felhasználók egyszerű listája mely mindösszesen az email címek eltárolása végett jött létre, kezdetben a jelszót is itt tárolta az alkalmazás, de biztonsági okok miatt ez megváltoztatásra került.

PageUserResults: Az eredményeket tároló tábla, melyben a felhasználó minden eredménye fellelhető. Szintén egyedi azonosítóval van ellátva minden rekord benne. Ezen kívül a játékokban szerzett egyes információk vannak eltárolva, mint a másodpercek, mely a játékban felhasznált másodpercet reprezentálja, a dátum, hogy mikor történt ez a játék, a rontási szám, amely számolja, hogy az adott játékban hány hibás szót írt be a felhasználó. Ezeken kívül eltárolódik a felhasználónak az email címe – mellyel történik a beazonosítása is – valamint a felhasználó szintje, mely egy lebegőpontos számban mutatható ki és utolsó sorban pedig egy szöveges visszajelzési forma, arról, hogy az adott játék a saját szintjéhez képest milyen kimenetelű volt: ez lehet pozitív, negatív, vagy semleges.

AspNetUsers: Az ASP-ben felhasznált biztonsági rendszer alaptáblája, ahol a felhasználó email címén kívül a jelszó van eltárolva, megfelelően elhashelve, hogy ne lehessen visszakövetni, kinyerni a nyers jelszót.

[20] DTO: Data Transfer Object, azaz adatátviteli objektum, melynek segítségével a webes felület könnyebben tud kommunikálni és adatot közvetítetni a szerver fele a végpontokon keresztül.

MultiGame: Az alkalmazás adatbázis szintjén megtalálható a korábban már lejátszott kétszemélyes játékok adatait eltároló MultiGame tábla. Az alappillére az azonosító, melyet felhasználva tudnak a felhasználók belépni egy már létező játékba. Ez egy hat karakterhosszú random generált azonosító, melynek minden párosadik karaktere egy számjegy. A két játékos email címe található meg a táblában, valamint két olyan mező amely azt zárja ki, hogy több játékos ne tudjon becsatlakozni, valamint a két játékosnak kirendelt szavak listáját tárolja el a végő elszámolás és eredményvisszajelzés végett.



21. ábra: Model réteg osztálydiagram

5.3 Logic réteg

A szerver oldala az alkalmazásnak tartalmaz egy logikai réteget, mely azért felelős, hogy az egyes számításokat és logikai műveleteket elvégezze, és külön részre bontsa az adatbázist és a Controller, vagyis az API endpoint réteget.

Mivel fontosnak tartottam, hogy ne létezzen egy bizonyos „God object[21]", ezért minden Controllerhez, egy számára dedikált és csak azt kiszolgáló Logic osztály került létrehozásra. A szavakat kezelő Logic osztály felel azért, hogy az adatbázisba betáplált szavak nehézségeit a matematikai képlet segítségével meghatározza és beállítsa.

MultiPlayerLogic: A kétjátékos mód logikáját és metódusait ellátó osztály, melyen keresztül tud a felhasználó létrtehozni, illetve becsatlakozni egy játékba. Az alkalmazás ezen osztályon keresztül nyeri ki a játékosok számára a szavakat, valamint a játék végén az adott eredményeket elmenti, illetve az adaptív pontozást tartva számítja ki a végső

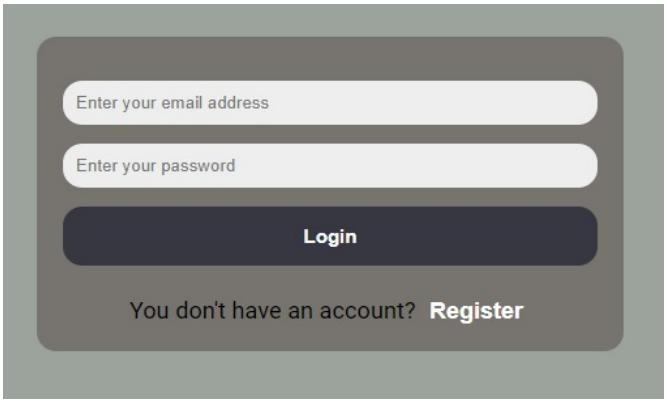
[21] God object: programozásban azt az osztályt szokás így nevezni, mely felelős mindenért egy alkalmazásban, rengeteg nem egymáshoz köthető metódussal és paraméterrel rendelkezik és nem követi az irányelveket és rétegzéseket

eredményt és menti el az adatbázisban. Ez az osztály fele a szobához tartozó egyedi azonosító legenerálásáért is.

5.4 Webes réteg, megjelenítés

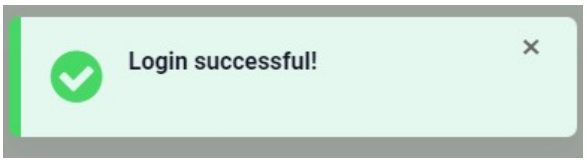
A webalkalmazás felépítése a különböző komponensekből áll, melyek lényegében egy-egy kifejezett feladatot látnak el.

Amikor a felhasználó elindítja az alkalmazást, egy bejelentkező, illetve regisztrációt biztosító felülettel találja szemben magát, ahogy azt a 22-es ábra is szemlélteti. Vendégfelhasználóként az oldal nem használható, hiszen fontos részét képezi az alkalmazásnak az eredmények tárolása és azoknak egyedi kiértékelése.



22. ábra: Bejelentkező, illetve regisztációs felület

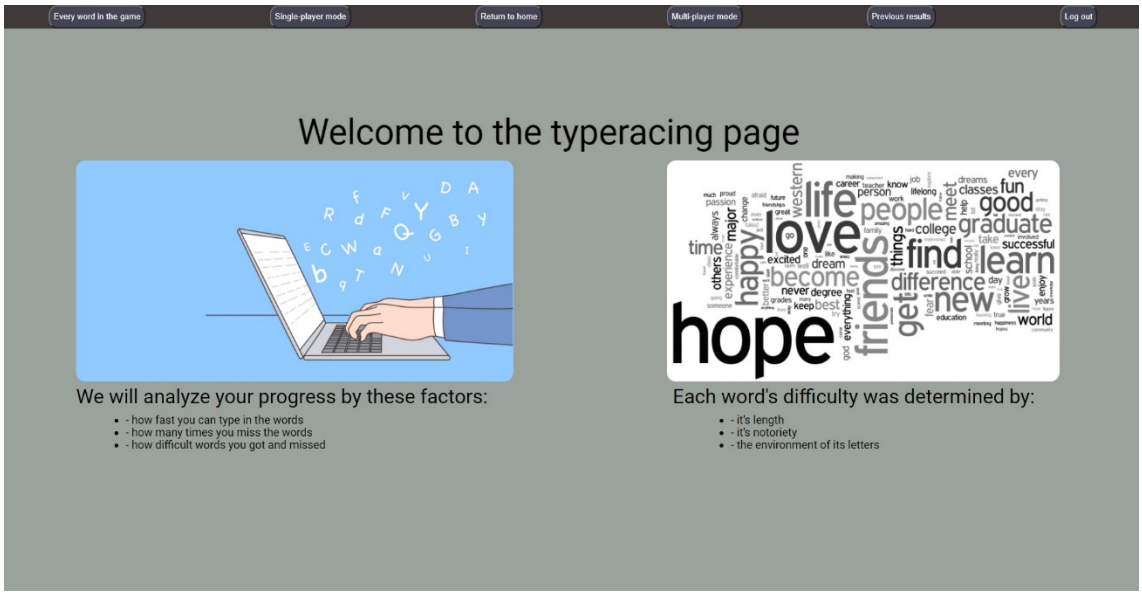
Amikor a felhasználó beírta az általa használt felhasználó nevet illetve jelszót, a szerver ezt ellenőrzi Bearer Authentikációval[22]. Visszajelzés végett a felhasználó egy értesítést kép a böngészőjének jobb felső sarkában, mely, amennyiben helyes email illetve jelszó párost adott meg, egy sikerességet kifejező zöld színben tűnik fel, amennyiben viszont valamilyen hibára futott az alkalmazás, egy tiltó piros jelzést kap a felhasználó. Ezeket az értesítéseket a Toastr[23] segítségével kapja meg a felhasználó, melynek példáját a 23-as ábrán szemléltetem, ahol sikeres bejelentkezés történt a felhasználó részéről.



23. ábra: Toastr visszajelzés sikeres bejelentkezés után

[22] Bearer Authentication: Másnéven token alapú authentikáció ami egy HTTP authentikációs séma, ami biztonsági tokenek segítségével működik, melyeknek a neve: bearer tokenek. Ezek a tokenek titkosított karaktersorozatok, melyeket a szerver generál a bejelentkezési kérelem válaszaként. A felhasználó bizonyos részeket csak ennek a tokennek a birtokában érhet el. [41]

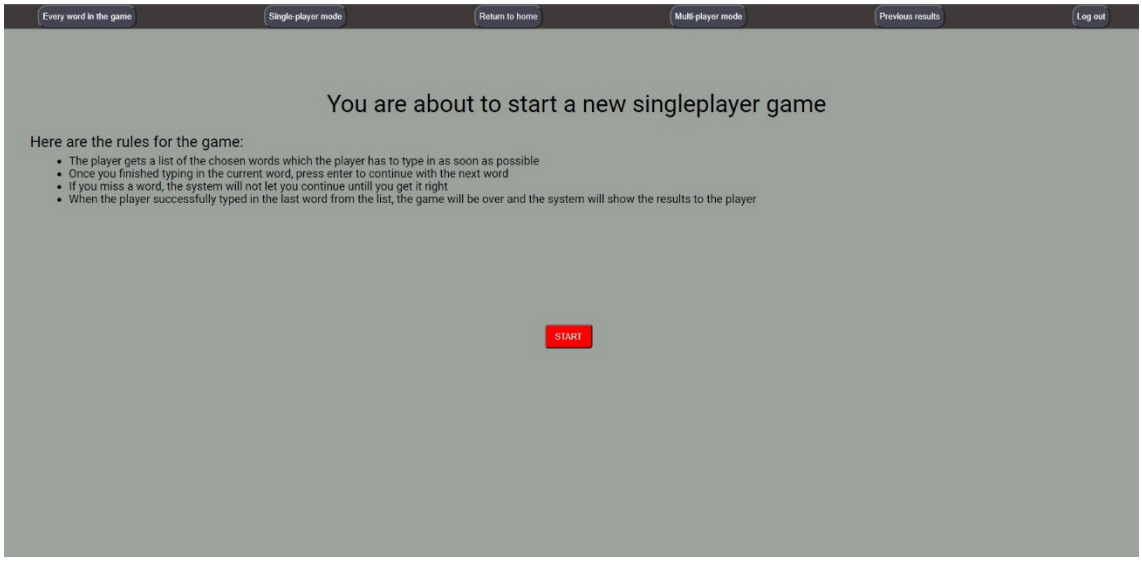
[23] Toastr: egy, nem blokkoló értesítéseket kínáló JavaScript könyvtár



24. ábra: Bejelentkezés utáni főoldal, ahol az oldal rövid ismertető jelei olvashatóak el

A sikeres bejelentkezést követően a felső sorban egy navigációs sáv van a felhasználó segítségére, melyen elhelyezkedő gombok használatával tudja az oldalt haszálni.

A felhasználónak lehetősége van egyszemélyes játékot indítania, melynél az oldalon röviden megjelennek a szabályok és az alap információk a játékos számára. Innentől márcsak egy gombnyomás választja el a játéktól.



25. ábra: Az egyjátékos mód közvetlen elindítási lehetősége, a piros háttérrel rendelkező Start feliratú gomb segítségével

Amikor a gombot a felhasználó megnyomta, az oldal automatikusan átirányítja egy új oldalra, ahol megjelennek a játékhoz szükséges elemek: egy stopper óra, mely az időt kezdi el mérni attól a pillanattól kezdve, amikor a felhasználó a gépelésre alkalmas mezőbe kattint és egy kilépést biztosító gomb.

A játék fő elemei, a szavak. Egyszerre csak egy darab szó olvasható ki az oldalon, ezt kell a felhasználónak legelőször legépelnie, majd ezután az Enter lenyomására a rendszer leellenőrzi, hogy megegyezik-e az elvárt és kapott szó minden karaktere, és abban az esetben, hogyha igen, a következő szó előbbre kerül. Egy hátralevő szavak visszaszámláló segíti a felhasználót abban, hogy tájékozódni tudjon afelől, hogy a játék mely szakaszában tart éppen, milyen eredménnyel.

26. ábra: Egyjátékos mód a játék legelején, ahol a felhasználónak az input mezőbe kattintása indítja el a stopperórát. A zöld betűszínnel jelzett oszlopban van az aktuális, leírandó szó, utána pedig elhomályosítva a következő szó

A játék betöltésekor egy API kérésen keresztül nyeri ki a rendszer a szükséges szavakat, melyeket eltárol, és egy kételemű listát alkalmazva mutatja ki egy keretnélküli táblázatban, mely jobbról haladva folyamatosan frissül a felhasználói interakció folyamán. Jól megfigyelhető az is, hogy az elhomályosított szó lesz valójában a következő szó minden esetben, így a felhasználó előre láthatja legalább a következő szónak a becsült hosszát.

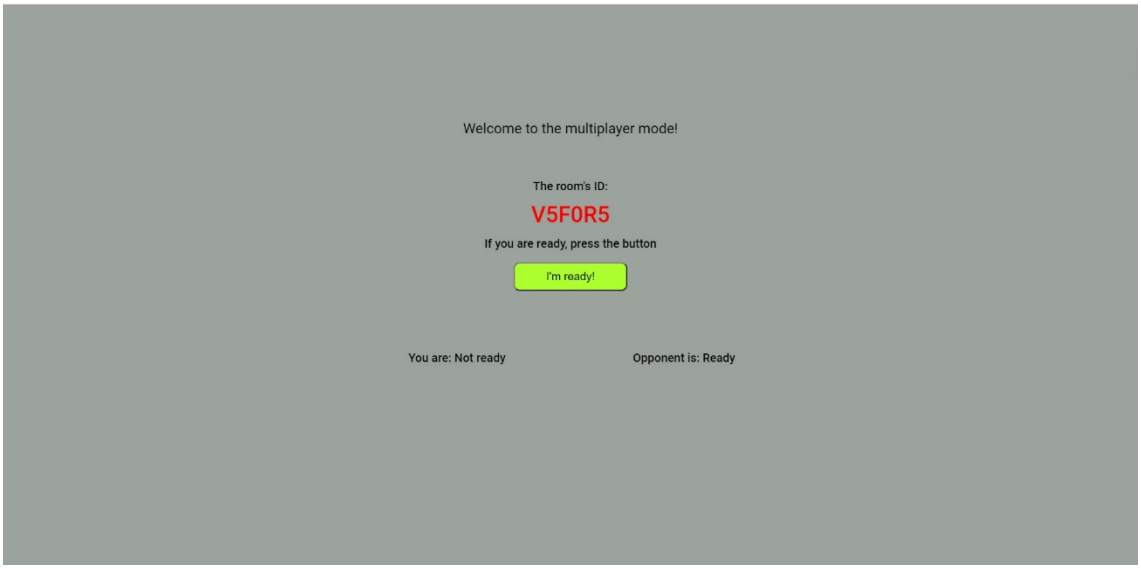
Az adott játékmenet csak abban az esetben mentődik el az adatbázisba, amennyiben a felhasználó az utolsó szót is helyesen begépelte. Egy szót akármennyiszer el tud rontani a játékos, a rendszer folyamatosan visszadobja neki egyből az elrontott szót, azaz addig nem haladhat tovább, ameddig az adott szó nincs helyesen leírva.

A felhasználónak bármelyik pillanatban lehetősége van abbahagyni a játékot, és a Quit gombbal kilépnie, ilyenkor a kezdő oldalra irányítódik vissza. Ebben az esetben viszont az eredmény nem kerül eltárolásra. Csak akkor mentődik el egy játék eredménye, és válik lekérdezhetővé, amennyibe a felhasználó az utolsó szót is sikeresen leírta és az oldal ilyenkor átirányítja egy másik oldalra ahol erről visszajelzést is kap a felhasználó.

A többjátékos mód kiválasztásánál a felhasználónak két opicó áll rendelkezésre: létrehozni egy új szobát, vagy egy már létező szoba egyedi azonosítójának beírására. Abban az esetben, ha a felhasználó hibás kódot ír be, a már bent tartózkodó játékos nem

kap értesítést, viszont egy hibajelzést a rossz kódot beíró felhasználó kap a rendszertől, de ettől függetlenül újrapróbálkozhat vele.

Mindkét esetben a felhasználó a kétjátékos módhoz tartozó várakozóban találja magát. Amennyiben ez a felhasználó hozta létre a szobát, abban az esetben a korábban már taglalt Toastr segítségével egy értesítést kap arról, hogyha egy másik játékos csatlakozott a szobához, és onnantól kezdve mindkét félnek lehetősége van arra, hogy jelezze elkészültségét a gomb lenyomásával. Ekkor, a szobában bent tartózkodó másik játékosnál az ellenfél jelző szöveg átváltozik, mindezek a signalr segítségével.

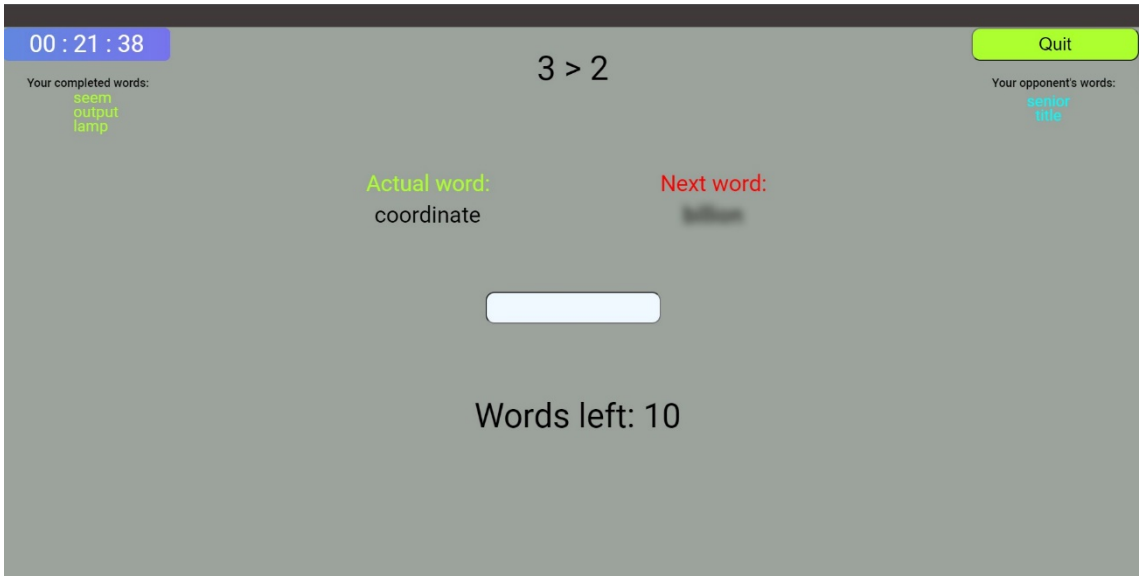


27. ábra: Kétjátékos mód előszoba, ahol pirossal van jelezve a szobának az egyedi azonosítója, melyet a becsatlazkoni kívánt játékos fele továbbítani kell. A gomb segítségével lehet jelezni a rendszer fele, hogy a felhasználó már készen áll, ezt abban az esetben, ha mind a két játékos belépett már a szobába

A teljes kétjátékos módot a signalr felügyeli a kapcsolatok terén. Mindkét felhasználó, egy úgynevezett csoportba lép be ennek segítségével, melynek neve kézenfekvő módon a szoba egyedi azonosítója. Amikor a játék sikeresen elindul, mindkét felhasználónál elindul a figyelés tevékenység, amely azt figyeli minden időpillanatban, hogy az adott csatornára érkezik-e új kész szó. Amennyiben igen, ezt a frontend lekezeli, és az ellenfél által teljesített szavakhoz kerül. Ugyanezen az elven elindulva, amikor egy felhasználó sikeresen legépelt egy szót, azt a csoport minden tagja fele – aki a kétjátékos mód miatt egyetlen főre korlátozódik – közvetíti, kivéve önmaga fele. Ezt a kikerülést a signalr-en belül azzal az opcióval oldottam meg, hogy minden felhasználó amikor csatlakozik egy ilyen csoporthoz, a saját connectionid-t használja, és amikor elküld egy új szót, akkor a csoport összes tagjának, kivéve önmagának küldi el, így kiküszöbölve azt az eshetőséget, hogy önmagát érzékelje a másik félnek.

Amikor mind a két játékos kijelentette, hogy készen áll, a rendszer automatikusan elindítja a játékot. A játék során a felhasználó számára a már ismert módszerrel jelennek meg a beírásra kapott szavak, és hogy hány darab van még hátra neki. Az egyjátékostól eltérően itt visszajelzést kap arról is a felhasználó egy relációs jel formájában, hogy a két

játékos közül ki áll nyerésre a kész szavak darabszámának függvényében. Ezen kívül, amikor a felhasználó beírt helyesen egy szót, az megjelenik számára a bal oldalon, mint kész szavak listája, ezen kívül, ha az ellenfél teljesített egy szót, akkor szintén a signalr segítségével ez is közvetítésre kerül, a képernyő jobb oldalán helyezkedik el az ellenfél által helyesen beírt szavak listája.



28. ábra: Kétjátékos mód játék közben, ahol a bal oldalon a felhasználó által már sikeresen beírt, a jobb oldalon pedig az ellenfél által sikeresen beírt szavak listája található meg, középen pedig egy helyzetjelentés afelől, hogy ez számbeli értékekben hogyan mutatkozik meg, és ki vezet – relációs jellel mutatva

Amennyiben az egyik játékos hamarabb végzett a szavaival, abban az esetben a másik játékos nincs befolyásolva, csak már nem kap újabb szavakat, melyet a másik teljesített.

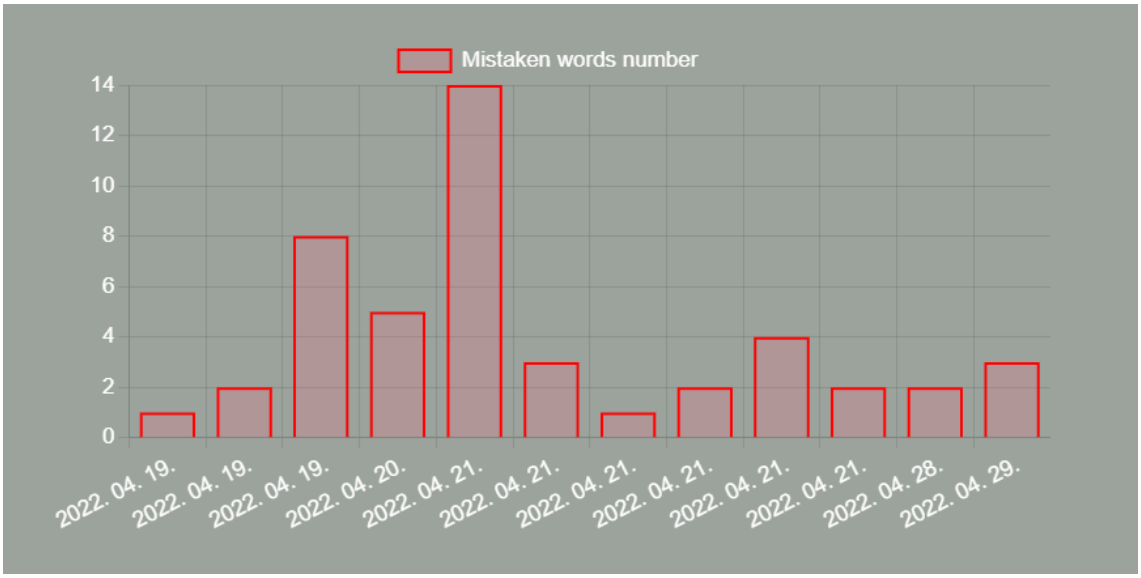
Fontosnak tartottam azt, hogy a böngészőben a felhasználónak ne legyen lehetősége szerkezteni az URL-t és manuálisan navigálnia magát az oldalak között – így akár egy számára korlátozott oldalt is el tud érni – így ezt az Angularban fellelhető Guard-ok segítségével oldotttam meg, melynek lényege, hogy minden komponenst egy bizonyos guard biztosít, melyben a logikát mi magunk írhatjuk meg. Az összes komponenst biztosító AuthGuard logikáját mindösszsen úgy építettem fel, hogy amennyiben a felhasználó kézzel próbálja meg átírni az URL-t, abban az esetben a sessionstorage[24]-ből a korábban elmenett tokent eltávolítjuk, és az AuthGuard azt ellenőrzi, hogy a felhasználó birtokában szerepel-e még a token. Amennyiben viszont a felhasználó fizikailag próbálja meg változtatni az URL-t, a token kitörlődik a tárolóból, és a rendszer így nem tudja átirányítani a beírt oldalra, ezáltal kilépteti a felhasználót a bejelentkező felületre.

A felhasználónak az oldalon keresztül számos lehetősége van arra, hogy a korábbi eredményeit lekérdezze. Erre szolgálnak a különböző részletes eredményeket kijelző diagrammok. Ezeket a diagrammokat a Chart.js[25] segítségével hozza létre az alkalmazás.

[24] Lokális böngésző tároló, mely a lap bezárásáig körübelül 5 mb-nyi adatot képes tárolni
[25] Chart.js: HTML5 alapú JavaScriptben levő diagrammok létrehozására szolgáló könyvtár

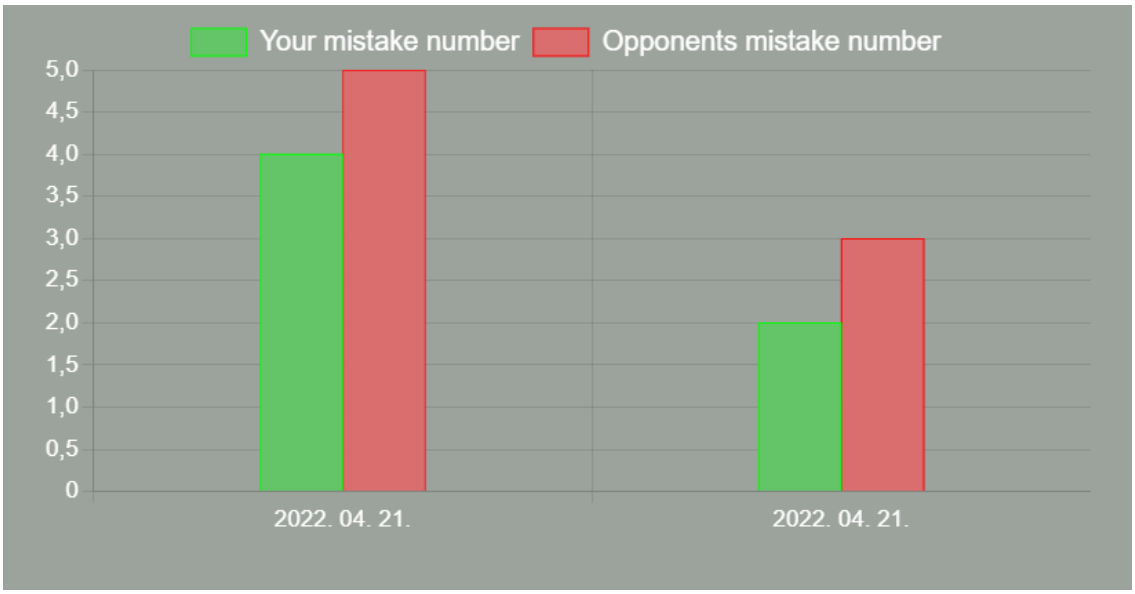
Az oldalon található „Previous results” fülre kattintva van lehetősége a felhasználónak megtekintenie a diagrammokat. Fontosnak tartottam azt, hogy a felhasználó által elért eredmények sokszínűségét és részletességét is kimutassák a diagrammok, így a felhasználó számára kilenc lehetőség áll rendelkezésre.

A 29-ik ábrán látható a három fajta diagram közül az első, egy oszlop diagram. Az idő függvényében olvasható ki, hogy a felhasználó az egyes játékai során – mind az egyjátékos, mind pedig a többjátékos módban – hány darab hibás szót írt le.



29. ábra: A felhasználó által elrontott szavak száma a dátum függvényében, ahol az oszlopok az elrontott szavak számát jelölik

A felhasználónak továbbá lehetősége van több adatot megjelenítő oszlopdiagrammot is lekérnie, melyen kizárólag azok a játékok eredményei szerepelnek, ami kétjátékos



30. ábra: A felhasználó kétjátékos módban rontott szavainak száma összehasonlítva az aktuális ellenfél rontott szavainak számával

módban történt. Ahogy azt a 30-ik ábra is szemlélteti, egymás mellett tűnik fel a felhasználó saját, illetve ellenfele hibásan beírt szavainak száma.

Egy átlagos nehézségi szinttel rendelkező szót leírni egy felhasználónak az adatok alapján körübelül hat másodpercig tart. A felhasználónak a szintjét kiszámító adaptív pontozás első pillérét ez által állítottam fel, miszerint a felhasználó tizenhárom darab szót kap egy játékban – függetlenül attól, hogy egyszemélyes, vagy kétszemélyes módról van szó – melynél így az idő értéke a képletben száz másodperc lesz.

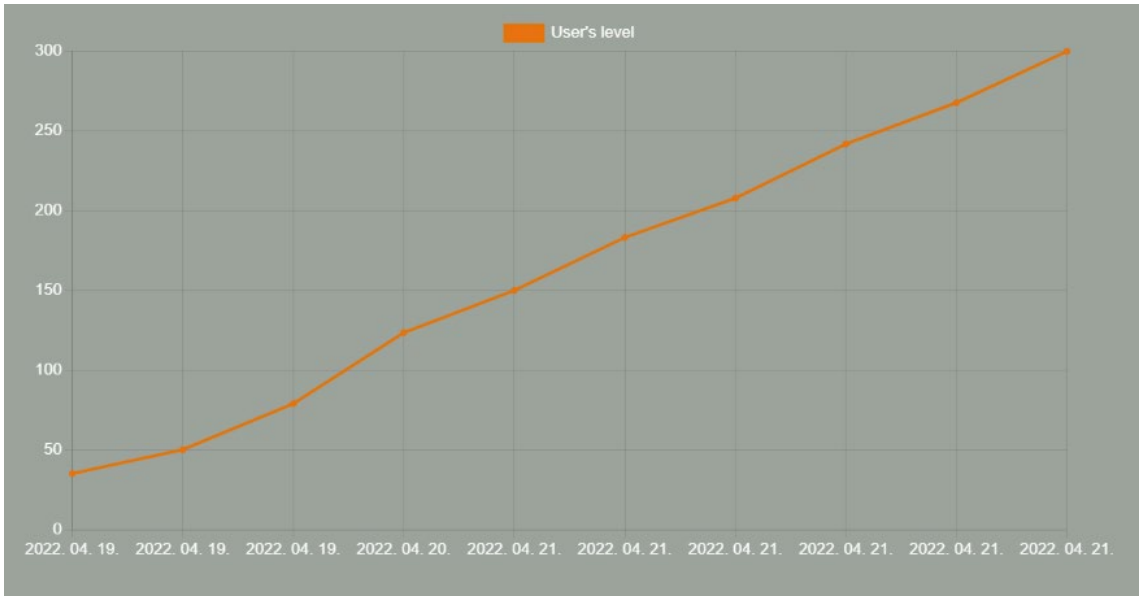
A második befolyásoló tényező, hogy a felhasználó milyen szavakat kapott. Egészen pontosan az befolyásol, hogy a kapott szavak milyen nehézségűek voltak. Minél nehezebbeket kapott, annál nagyobb legyen az érték. Ehhez meg kellett határozni, és figyelembe kellett venni, hogy az adatbázisban fellelhető összes szónak milyen az átlagos nehézsége, és azzal elosztani a szumma értéket.

Amikor a felhasználónak a tapasztalati szintjét számítjuk, fontos még megvizsgálnunk az adott játékban elrontott szavak eloszlását és azoknak nehézségeit, hiszen minél több volt, annál rosszabbul teljesített.

$$\frac{100\ (mp)}{eltelt\ idő\ (mp)} \bullet \frac{kapott\ nehézség}{átlag\ nehézség} - \frac{elrontott\ szavak\ nehézsége}{elrontott\ szavak\ száma}$$

31. ábra: Felhasználói tapasztalati szintet kiszámító adaptív pontozási képlet

A felhasználó tapasztalati szintjét a képlet után kapott eredménnyel növeli a rendszer.

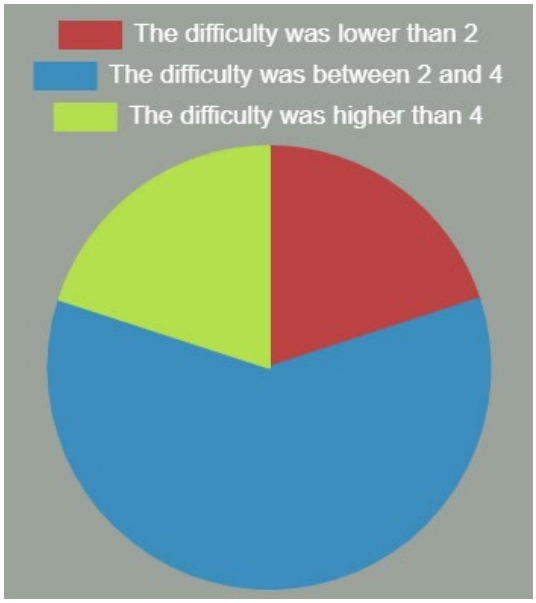


32. ábra: A felhasználó tapasztali szintjének előrehaladását kimutató diagram a játszmák időpontjának függvényében

Azonban a szöveges visszajelzést, miszerint az adott játékban elért eremédménye:

pozitív, negatív vagy semleges eredmény volt, befolyásolja továbbá az is, hogy a felhasználónak milyen korábbi eredményei voltak. Ezen meghatározáshoz, az fog számítani, hogy a felhasználónak az eddigi összes meccse alapján milyen átlagértéket ért el a tapasztalati szintje mellett. Amennyiben ennél az értéknél jobbat tudott produkálni, abban az esetben pozitív lesz a végső konklúzió, azonban ha egy konstans értéknél nem ér el többet a felhasználó egy játék alatt, akkor az negatívként mentődik el az adatbázisban.

A Chart.js-nek köszönhetően az elkészített diagrammok reszponzívak, azaz nem csak megjelennek, hanem minimálisan plusz információt is képesek szolgáltatni. Erre a legjobb példa a 34-ik ábrán látható kördiagram, ahol a felhasználónak lehetősége van arra is, hogy az egyes eredményt megsemmisítse, és anélkül tekintse meg a diagrammot



33. ábra: Kördiagram eloszlása a szavak nehézségének függvényében, ahol kék színnel jelölt terület azon szavak mennyisége amelyeknek a nehézsége 2 illetve 4 között volt, piros színnel amelyeknek kisebb volt az értéke mint 2, végül a sárga szín jelöli azon szavakat amelyeknek a nehézsége nagyobb volt mint négy.

6. Tesztelés

A tesztelés egy rendszer kontrollált körülmények melletti futtatása, és az eredmények kiértékelése, mely több részre bontható. A szakdolgozatomban két részre bontottam fel a teszt eseteket: backend illetve frontend. A backend oldali metódusok és osztályok tesztelésére Unit Test[26]-eket írtam, míg a frontend részen tesztelhető komponenseket és API végpont hívásokat a Postman[27] nevezetű programmal valósítottam meg.

6.1 Backend oldal tesztelése

A Unit tesztek segítségével a backend oldal tesztelése elszeparált módon tud történni, és egyesével tesztelhetőek – láthatósági és hozzáférhetőségi szinttől függően – az egyes metódusok, osztályok és fájlkezelések működései.

6.1.1 Egy, illetve többjátékos módhoz tartozó tesztek

#	Tesztleírás	Mérés és cél	Eredmény
1.	A felhasználó számára a megfelelő darabszámú szólista lekérése	Az adatbázisról a szükséges darabszámú lista kinyerése random kiválasztással	
2.	A felhasználó kétjátékos módba való beléptetése hibás szoba azonosítóval	Error üzetet kapása, miszerint nem létezik ilyen szoba	
3.	Egyedi szobaazonosító generálása, melynek a hossza hat karakter	Egy hat karakterből álló egyedi karakterlánc	
4.	A felhasználó eddigi játszott összesítése egy számbeli értékben	Email cím alapján való keresés az adatbázisban	
5.	A felhasználó legutolsó elért tapasztalati szintje	Lebegőpontos számértékben a felhasználó tapasztalati szintjének kinyerése	
6.	Egy egyedi azonosítóval ellátott kétszemélyes szoba létrehozása	Az adatábázisban egy új kétjátékos szoba legenerálása	
7.	Egy már létező kétszemély módba való becsatlakozás	Az egyedi azonosítót beírva egy már létező szobába való becsatlakozás	

34. Ábra: Az egyes játékokhoz tartozó tesztesetek leírása, célja és eredménye

[26] Egy egység külön, elszigetelt módon való tesztelése.
[27] Egy könnyen használható API platform, leginkább a már létező végpontok tesztelésére szoktak használni.

6.1.2 Eredmények lekérdezésére szolgáló tesztek

#	Tesztleírás	Mérés és cél	Eredmény
8.	Email cím szerinti adatbázisban fellelhető eredmények lekérdezése	Megfelelő méretű lista kinyerése az adatbázisból	
9.	Az ellenfél megtalálása a másik email címe és a szoba azonosítójának birtokában	Szoba azonosító és email cím esetén az adatbázisból az email cím kinyerése	
10.	Email cím szerint fellelhető összes eredmény lekérdezése	Megfelelő méretű lista kinyerése az adatbázisból	

35. Ábra: A különböző felhasználói eredményeket lekérdező metódusok tesztelésének eredménye

6.1.3 Szavak nehézségének meghatározásához tartozó tesztek

#	Tesztleírás	Mérés és cél	Eredmény
11.	Egy karakter környezetének megtalálása	Megfelelő környezet meghatározása a karakternek	
12.	Az adatbázisba való szavak sikeres feltöltése	A feltöltött szavak száma megegyezik az előre meghatározottal	

36. Ábra: A szavak egyedi nehézségének meghatározásához tartozó metódusok tesztjei illetve eredményük

6.2 Frontend oldal tesztelése

A frontend oldali teszteket, vagyis az API végpontok tesztelését a Postmannal oldottam meg. Az egyes végpontok – melyeket a controller osztályok tartalmaznak – egyesével tesztelhetőek.

6.2.1 Bejelentkezéshez tartozó végpontok tesztelése

#	Tesztleírás	Mérés és cél	Eredmény
1.	A felhasználó email címe és jelszavával való bejelentkezés tesztelése	A végpont meghívása után egy token és lejárati dátum visszakapása eredményként	
2.	Az oldalra regisztrált összes felhasználó adatainak lekérdezése	Biztonsági protokollokat betartva az információk listázása	
3.	Egy új felhasználó regisztrálása az oldalra és az adatbázisba	A jelszó elhashelt értékkel való elmentése az adatbázisba	

37. Ábra: Bejelentkezés, regisztrálás illetve egy listázó lekérdezés tesztelése az API végpontokon keresztül

6.2.2 Adatbázist kezelő és segítő végpontok tesztelése

#	Tesztleírás	Mérés és cél	Eredmény
4.	A szavak tábla feltöltése a szöveges dokumentumokból	Az összes szó nehézségének mérése és adatbázisba való elmentése	
5.	A szavak tábla rekordjainak redukálása az esetleges duplikációk végett	Az ismétléses szavak kiszűrése és az adatbázis frissítése	
6.	Hiba esetén lehetőség az adatbázisban található összes szó eltávolítására	Az összes szó eltávolítása az adatbázisból a tábla megőrzése mellett	

38. Ábra: Az adatbázis szavak táblájához tartozó feltöltés, törlés, illetve redukálás tesztelése az API végpontokkal

6.2.3 Egyjátékos módot kezelő végpontok tesztelése

#	Tesztleírás	Mérés és cél	Eredmény
7.	A felhasználónak egy szavakat tartalmazó lista kinyerése	Megfelelő méretű szólista előállítása a felhasználó számára	
8.	A felhasználó eredményének elmentése az adatbázisba	DTO segítségével a paraméterek átadása és az adatbázis frissítése	

39. Ábra: Az egyjátékos módban felhasznált szólista generálása illetve az eredmény elmentésének tesztelése

6.2.4 Többjátékos módot kezelő végpontok tesztelése

#	Tesztleírás	Mérés és cél	Eredmény
9.	Kétszemélyes módhoz tartozó privát szoba létrehozása	Szoba létrehozása egyedi azonosítóval és annak mentése az adatbázisba	
10.	Kétszemélyes módhoz tartozó privát szobába való belépés	Már létező, adatbázisban is szereplő szobába való csatlakozás	
11.	Felhasználó számára a szólista kinyerése az adatbázisból	Véletlenszerű szavakat tartalmazó, fix méretű lista kinyerése	
12.	A játék végén elért eredmény elmentése az adatbázisba	Az adatbázis megfelelő táblájának frissítése az új rekord elhelyezése után	

40. Ábra: Kétjátékos módhoz tartozó szobát, játékot és eredményeket kezelő végpontok tesztelése

6.2.5 A diagramokhoz tartozó végpontok tesztelése

#	Tesztleírás	Mérés és cél	Eredmény
13.	A többjátékos módban elrontott szavakat kimutató diagramhoz való eredmény	A lekérdezés eredményének tesztelése az adatbázissal összevetve	
14.	A felhasználóhoz tartozó összes eredmény lekérdezése az adatbázisból	Email cím alapján való lekérdezés, mely egy lista formájában tér vissza	
15.	A felhasználó és az ellenfele együttes eredmény lekérdezésének tesztelése	Lista formájában az eredmény közlése ahol a felhasználó más ellen játszott	

41. Ábra: Az oldalon található diagramokhoz tartozó segéd végpontok tesztelése a diagramok pontos működése érdekében

A Postman segítségével megvalósíthatóak mind a GET, POST illetve PUT kérések is, melyek szükségesek voltak az API végpontok teszteléséhez.

- Get kérést akkor használtam, amikor az adatbázist nem módosító és authentikációs logikát nem tartalmazó végpontokat teszteltem, például a felhasználók listájának lekérdezése, vagy a játékhoz tartozó szólista lekérdezése.
- Post kérést azokban az esetekben alkalmaztam, amikor az adatbázis egyes tábláiba új rekordokat illesztettem be, ilyen volt például egy új felhasználó regisztrálása, vagy éppen egy felhasználó új eredményének elmentése.
- Put kérést csak akkor hajtottam végre, amikor az adatbázisban már szereplő rekordot szerettem volna módosítani, vagy olyan logikát kívánt meg, melyet a Get kérésben már nem lehetett teljesíteni, ilyen volt például a felhasználó bejelentkeztetése, vagy egy szoba létrehozása, illetve abba való becsatlakozása.

7. Értékelés

Az irodalomkutatásom során megvizsgált hasonló rendszereket alapul véve, és azoknak a hiányosságait kiküszöbölve fejlesztettem ki a webalkalmazásomat. Figyelembe kellett vennem azokat a kiegészítéseket, melyeket a megvizsgált rendszerek nem teljesítettek, például az adaptív pontozásra, illetve az egyes szavak nehézségének meghatározására.

Az alábbi táblázatban próbáltam szemlélteni az eltéréseket a korábban már megvizsgált és a saját rendszerem között:

Elvárás a rendszerrel szemben	Typeracer	Typerush	Saját rendszer
Kapott szavak fontossága			
Egyjátékos mód lehetősége			
Kétjátékos mód lehetősége			
Több mint két játékos mód lehetősége			
Eredmények elmentése és lekérdezhetősége			
Eredmények közlése diagramok ábrázolásával			
Kötelező regisztráció/bejelentkezés a felhasználónak			
Felhasználó szintjének adaptív számítása			
Szavak nehézségének egyedi meghatározása			

Ahogy a táblázatból jól kiolvasható, a saját rendszeremben az eredmények, és az adaptivitáson volt a hangsúly. Míg a többi rendszerben a kapott szavaknak a nehézsége nem számot tevő, a saját rendszeremben egy jelentős hangsúlyt került erre a területre.

Míg a megvizsgált rendszerek legfeljebb – amennyiben regisztrált felhasználóról van szó – annyit biztosítanak korábbi eredményközlés végett a felhasználó felé, hogy milyen szavakat kapott, a saját rendszeremben számos diagram szolgál az eredmények közlésére, és a felhasználó teljesítményének visszajelzésére.

A korábban, a tervezés fázisban meghatározott funkciólistán szereplő funkciók jelentős része teljesült a saját rendszeremben, miszerint a szavak nehézsége egyedi módon kerül kiszámításra, épp úgy, ahogyan a felhasználó tapasztalati szintjének növekedése vagy csökkenése is adaptív pontozási rendszert használ.

A tervezés fázisban meghatározásra került két fajta játékmód melyek közül a felhasználó választhat, mind a kettő sikeresen meg lett valósítva, ahogy a tervezésnél részletezésre került.

A Typeracer illetve Typerush rendszerekkel szemben, a saját rendszeremben nem került sorra a fejlesztés során az a lehetőség, hogy a harmadik játékmód is elérhető legyen a felhasználónak, ami a többjátékos mód lett volna. Ebben az esetben egy szobában a létszám nem korlátozódott volna le két főre, hanem akár négy játékos is egymás ellen játszhatott volna, a képernyőt felosztva a részeredményekkel.

A fentebb már említett rendszerekhez képest a saját rendszerem kevesebb felhasználói élményt fokozó funkcióval rendelkezik, viszont ezt kompenzálja a sokszínű eredményközlés lehetősége, miszerint a felhasználónak tíz opció közül van lehetősége választani aszerint, hogy a korábbi teljesítményei mely részéről szeretne részletesebb visszajelzést kapni, mindezt oszlop, vonal, vagy éppen kördiagram segítségével.

8. Továbbfejlesztési lehetőségek

Ahogyan az értékelés fejezetben említettem, az elmaradt és nem megvalósított funkció, melyet a rendszerben meg lehetne valósítani az a több mint kétjátékost kiszolgáló lehetőség. Egyrészt azért, mert a kétjátékos mód korlátozza a lehetőségeket és az együttjátszhatóság elvet, másrészt mert ha több mint két fő szeretne játszani – nem egyedül – abban az esetben így egy embernek várakoznia kell.

Továbbfejlesztési lehetőségként lehet a felhasználói élmények növelésére szolgáló funkciókra is tekinteni, mint például egy jutalmazási rendszer, ahol a felhasználó minél többet használja a rendszert és ezzel együtt, minél jobb eredményeket is ér el, annál nagyobb jutalmat kap, például egy virtuális összeg keretein belül, ahol ezt az összeget, az oldalon fel is tudja használni olyan dolgokra, mely akár az oldalon levő kinézetét tudja alakítani, vagy akár privát, lezárt funkciók elérése is feljogosíthatja magát, például: Sötét mód alkalmazása az oldalon.

A megvizsgált rendszerekben fellelhető, hogy a többjátékos mód során, a játékost egy ikon ábrázolja, és ezt a felhasználó tudja változtatni, és a többi játékos virtuális, ikonbeli kinézete is látható a többiek számára, egy továbbfejlesztési opció lehet ebben a területben mélyebbre ásni, és a változtatás jogát a felhasználónak adni.

Az általam fejlesztett rendszer a fentebb említett továbbfejlesztési lehetőségekkel számos területen és közösségben tudna érvényesülni:

- Iskolásoknak, akik az angol idegennyelvet tanulják, azzal az opcióval, hogy lehetőség lenne számukra beállítani egy saját adatbázist, ahonnan kerülnek kiválasztásra a szavak, így gyakorolva a nyelvet
- Azon embereknek, akik szeretnének megtanulni minél gyorsabban pontosan gépelni
- Bármely olyan oktatási célcsoportnak akiknek valamilyen kompetenciát mérő feladat kell

Irodalomjegyzék

[1] [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/How_the_Web_works. [Hozzáférés dátuma: 13 11 2021].

[2] „TutorialsPoint,” [Online]. Available: https://www.tutorialspoint.com/asp.net/asp.net_introduction.htm. [Hozzáférés dátuma: 14 11 2021].

[3] „MDN Web Docs,” [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/MVC. [Hozzáférés dátuma: 15 11 2021].

[4] A. Lock, ASP.NET Core in Action, Second Edition.

[5] „Nyelvek - info elte,” [Online]. Available: http://nyelvek.inf.elte.hu/leirasok/ASP.NET/index.php?chapter=1#section_1. [Hozzáférés dátuma: 16 11 2021].

[6] [Online]. Available: https://www.tutorialsteacher.com/javascript/prototype-in-javascript. [Hozzáférés dátuma: 01 12 2021].

[7] J. N. Robbins, Learning Web Design.

[8] [Online]. Available: http://www.inf.u-szeged.hu/~tarib/javascript/. [Hozzáférés dátuma: 23 11 2021].

[9] W. D. f. Beginners, White Belt Mastery.

[10] Sipos Miklós, JavaScript - Theory ° Practice, 2021.

[11] „MDN Web Docs,” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. [Hozzáférés dátuma: 01 12 2021].

[12] „W3schools,” [Online]. Available: https://www.w3schools.com/js/js_htmldom.asp. [Hozzáférés dátuma: 01 12 2021].

[13] „nodeJS,” [Online]. Available: https://nodejs.dev/learn/modern-asynchronous-javascript-with-async-and-await. [Hozzáférés dátuma: 01 12 2021].

[14] „Learn Node js,” [Online]. Available: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm. [Hozzáférés dátuma: 24 11 2021].

[15] „NetGuru,” [Online]. Available: https://www.netguru.com/glossary/node-js. [Hozzáférés dátuma: 24 11 2021].

[16] SimpliLearn. [Online]. Available: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs. [Hozzáférés dátuma: 20 11 2021].

[17] „NetGuru," [Online]. Available: https://www.netguru.com/glossary/node-js. [Hozzáférés dátuma: 23 11 2021].

[18] „TutorialsPoint - typescript," [Online]. Available: https://www.tutorialspoint.com/typescript/typescript_overview.htm. [Hozzáférés dátuma: 24 11 2021].

[19] „Khan Academy," [Online]. Available: https://www.khanacademy.org/computing/computer-programming/html-js-jquery/jquery-intro/v/what-is-jquery. [Hozzáférés dátuma: 27 11 2021].

[20] „TechTarget - Search Data Management," [Online]. Available: https://searchdatamanagement.techtarget.com/definition/MongoDB. [Hozzáférés dátuma: 27 11 2021].

[21] „D3 - A Beginner's Guide to Using D3," [Online]. Available: https://website.education.wisc.edu/~swu28/d3t/. [Hozzáférés dátuma: 27 11 2021].

[22] „DBpedia," [Online]. Available: https://dbpedia.org/page/TypeScript. [Hozzáférés dátuma: 27 11 2021].

[23] „TutorialsPoint," [Online]. Available: https://www.tutorialspoint.com/typescript/typescript_overview.htm. [Hozzáférés dátuma: 27 11 2021].

[24] „Techopedia," [Online]. Available: https://www.techopedia.com/definition/24580/intellisense. [Hozzáférés dátuma: 27 11 2021].

[25] „Tutorialspoint," [Online]. Available: https://www.tutorialspoint.com/typescript/typescript_overview.htm. [Hozzáférés dátuma: 27 11 2021].

[26] „Serokell," [Online]. Available: https://serokell.io/blog/why-typescript. [Hozzáférés dátuma: 27 11 2021].

[27] „DevelopIntelligence," [Online]. Available: https://www.developintelligence.com/blog/2016/05/angular-bootstrap-explained-3-examples/. [Hozzáférés dátuma: 27 11 2021].

[28] „Angular," [Online]. Available: https://angular.io/guide/architecture. [Hozzáférés dátuma: 27 11 2021].

[29] O. Campesato, Angular and Deep Learning - Pocket Primer.

[30] „Dummies - programming,” [Online]. Available: https://www.dummies.com/programming/what-does-css-do/. [Hozzáférés dátuma: 27 11 2021].

[31] „Socket.IO,” [Online]. Available: https://socket.io/docs/v4/. [Hozzáférés dátuma: 01 12 2021].

[32] „ably,” [Online]. Available: https://ably.com/topic/socketio. [Hozzáférés dátuma: 01 12 2021].

[33] „ResearchGate,” [Online]. Available: https://www.researchgate.net/figure/Average-word-length-in-the-English-language-Different-colours-indicate-the-results-for_fig1_230764201. [Hozzáférés dátuma: 28 11 2021].

[34] „Oxford Learner's Dictionaries,” [Online]. Available: https://www.oxfordlearnersdictionaries.com/about/wordlists/oxford3000-5000. [Hozzáférés dátuma: 28 11 2021].

[35] „Computerhope,” [Online]. Available: https://www.computerhope.com/issues/ch001346.htm. [Hozzáférés dátuma: 28 11 2021].

[36] „WebShark,” [Online]. Available: https://webshark.hu/hirek/html/. [Hozzáférés dátuma: 01 12 2021].

[37] „MiMi,” [Online]. Available: https://www.mimi.hu/informatika/dom.html. [Hozzáférés dátuma: 01 12 2021].

[38] „W2schools,” [Online]. Available: https://www.w3schools.com/js/js_callback.asp. [Hozzáférés dátuma: 01 12 2021].

[39] „W3schools,” [Online]. Available: https://www.w3schools.com/js/js_promise.asp. [Hozzáférés dátuma: 01 12 2021].

[40] [Online]. Available: https://www.tutorialspoint.com/asp.net/asp.net_introduction.htm. [Hozzáférés dátuma: 15 11 2021].

[41] „DevOpsSchool,” [Online]. Available: https://www.devopsschool.com/blog/what-is-bearer-token-and-how-it-works/. [Hozzáférés dátuma: 19 04 2022].

Ábrajegyzék