ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Pap Tamás**
2022	Hallgató törzskönyvi száma:	**T/006644/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Pap Tamás**
Törzskönyvi száma: T/006644/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Online platform programozási párharcokhoz és önálló gyakorláshoz
Online platform for coding battles and self-practicing

Intézményi konzulens: Kovács András, Gáspár Balázs
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Szoftvertechnológia és grafikus felhasználói interfész tervezése
Szoftvertervezés és -fejlesztés specializáció

A feladat

A hallgató feladata egy olyan webalkalmazás tervezése és fejlesztése, ahol a felhasználók egymás ellen tehetik próbára programozási készségüket.
Az alkalmazás rendelkezzen egy- és kétjátékos üzemmóddal. Ebből előbbi az önálló gyakorlást szolgálja, míg utóbbi esetben két felhasználónak egy időben, ugyanarra a programozási feladványra kell saját megoldást készítenie. A nyertes, aki hamarabb oldja meg helyesen a feladatot.
Az alkalmazás biztosítson egy felületet, ahol a felhasználók megadhatják megoldásaikat egy adott feladatra. A rendszernek képesnek kell lennie a felhasználók által írt kódokat futtatni és tesztelni, ezek eredményeiről visszajelzést küldeni. Az alkalmazás által kezelt programozási nyelv a hallgató által tetszőlegesen választható. A felhasználóknak legyen lehetőségük regisztrálni az oldalra, a rendszer mentse eredményeiket.

A dolgozatnak tartalmaznia kell:

- a kapcsolódó szakirodalom áttekintését, a hasonló rendszerek rövid bemutatását,
- az elkészítendő alkalmazás pontos specifikációját,
- a részletes rendszertervet,
- a megvalósításhoz használt technológia választásának indoklását,
- a szoftver implementálását, a fejlesztés részletes dokumentációját,
- a felhasználói dokumentációt,
- a továbbfejlesztési lehetőségek bemutatását.

P.H.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 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

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
7. sz. melléklet

HALLGATÓI NYILATKOZAT

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

Budapest, 2022.05.12

.....
hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun Kód:	Tagozat:
Pap Tamás	[redacted]	NAPPALI

Telefon: Levelezési cím (pl.: lakcím):

[redacted]

Szakdolgozat címe magyarul:

Online platform programozási párharcokhoz és önálló gyakorláshoz

Szakdolgozat címe angolul:

Online platform for coding battles and self-practicing

Intézményi konzulens: Külső konzulens:

Kovács András Gáspár Balázs

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 09. 20	Téma tervezése	
2.	2021. 10. 18	Irodalomkutatás ellenőrzése	
3.	2021. 11. 15	Rendszer tervezése	
4.	2021. 12. 10	Végső ellenőrzés, prezentációs próba	

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.

.....

Intézményi konzulens

Budapest, 2021. 12. 10

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun Kód:	Tagozat:
PAP TAMÁS	[redacted]	NAPPALI

Telefon: Levelezési cím (pl.: lakcím):

[redacted]

Szakdolgozat címe magyarul:

ONLINE PLATFORM PROGRAMOZÁSI PÁRHARCOKHOZ ÉS ÖNÁLLÓ GYAKORLÁSHOZ

Szakdolgozat címe angolul:

ONLINE PLATFORM FOR CODING BATTLES AND SELF-PRACTICING

Intézményi konzulens: Külső konzulens:

KOVÁCS ANDRÁS, GÁSPÁR BALÁZS

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.15.	IMPLEMENTÁCIÓ TERVEZÉS	
2.	2022.03.25.	IMPLEMENTÁCIÓ ÁTTEKINTÉSE	
3.	2022.04.10.	TESZTELÉSEK KIÉRTÉKELÉSE	
4.	2022.05.10.	VÉGSŐ ELLENŐRZÉS, LEZÁRÁS	

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. május 10.

Tartalom

A szakdolgozat alatt használt kifejezések és ezeknek a jelentései

Kifejezés	Jelentés
Backend	Szerveroldalon futó alkalmazás
Frontend	Kliensoldali alkalmazás
API	Application Programming Interface
REST	Representational State Transfer
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
JSX	JavaScript XML
XML	Extensible Markup Language
DOM	Document Object Model
JWT	JSON Web Token
JSON	JavaScript Object Notation
Repository	Általános tároló

1. Bevezetés

A mai világban a tapasztalt programozóknak sok nyelvel van tapasztalatuk, sok problémára tudják már az optimális megoldásokat. Ezzel ellentétben vannak azok az emberek, akik még nem írták meg az első „Hello World” alkalmazásukat, de ki szeretnék magukat próbálni. Erre a kettő szélsőség és a közöttük lévő embereknek jól jöhet egy oldal, ahol akár napi pár percben, de tudják fejleszteni magukat, és nem kell egy projektbe belekezdeni. Erre a problémára adhat egy megoldást a szakdolgozat feladata, ami mind gyakorlásra, mind pedig mások elleni játékara alkalmas. A feladat fog rendelkezni különféle játékmódokkal, például privát és publikus két játékos móddal. Ez azoknak jelenthet majd megoldást, akik szeretnének mások ellen játszani, szórakozás közben elsajátítani új optimális megoldásokat. Igaz a feladat biztosít majd egy szövegmezőt a játékosoknak, ahová tudnak majd kódolni, de mivel a feladatoknak lesz egy leírása példa bementtel és eredménnyel, ezért nem lesz kötelező a weboldal által biztosított szövegmezőt használnia a felhasználónak, hanem megírhatja a megoldását a saját, már jól betanult fejlesztő környezetében, majd, ha végzett és szeretné a kódot beküldeni megoldásnak, akkor csak a weboldal szövegdobozába másolja. A szakdolgozat első felében be fogom mutatni, hogy egy webalkalmazás elkészítését hogyan lehet elkezdeni. Be lesz mutatva a Backend és Frontend ma használt talán legnépszerű keretrendszerei. Felvetésre kerül a probléma, hogy hogyan tudunk egy olyan funkciót létrehozni, hogy a többjátékos módban játszó játékosok kommunikálni tudjanak egymással. Erre a problémára a WebSocket lesz bemutatva, mint egy megoldás. Részletezem a népszerű Frontend keretrendszereket, az Angular, React és Vue.Js-t. Backend oldalon három rendszer lesz leírva, mind különböző nyelven íródott, de funkcionalitásukban szinte megegyeznek majd. Ezek lesznek a NestJs, Laravel és Django. Ezen felül bemutatásra kerül két hasonló weboldal, ami a szakdolgozat témájához talán a legjobban hasonlítható, így jobban megértve a dolgozat célját. Ezt fogja követni egy fejezet arra, hogy a projektnek pontosan mit kell elérni, mint frontend mint backend oldalról. Mivel egy fejlesztés menete sokkal könnyebb lehet azzal, hogy előre megtervezzük az egységeket, különböző típusú diagrammokkal ábrázoltatjuk, egy kisebb egységekben ott pont mi mit követ, ezért egy külön fejezetben ezt be fogom mutatni. Itt szó fog esni egy játékmódnak a kiválasztásáról, ezeknél mi kell, hogy történjen egy játék elindulásához, szó lesz arról, hogy amennyiben már megy a verseny, úgy milyen lépések futnak le aközött, hogy egy

játékos megnyomja a futtatás gombot, és hogy ezeknek az eredményéről egy értesítést kapjon. Ezek után jön egy külön fejezet arra, hogy a fejlesztés hogyan zajlott. Mint ahogy pár más helyen, itt is külön lesz választva a Backend és a Frontend rész. Még mielőtt ezekről szó esne, be lesz mutatva hogyan lehet egy projektet létrehozni, milyen parancsokat kell futtatni ahhoz, hogy üres mappából egy 'monorepo' keletkezzen. Backend oldalról csak fő funkciók alapjai lesznek, hiszen rengeteg apró kis elemből épül fel a projekt, és ezt nagyon hosszan lehetne sorolni, hogy mi, miért és hogyan lett felhasználva. Frontend oldalról főleg a három fő weboldal felépítése lesz, hogyan néznek ki, milyen fő funkciók találhatóak meg rajta, kódi szinten pár elemet hogyan sikerült működésre bírni. Szó esik majd a fejlesztés közben előfordult főbb hibákról, ezeket mi okozta és mi volt rájuk a megoldás. Mivel szeretném, hogy a projekt elérhető legyen bárki számára, ezért egy kis bekezdésben azt is meg fogom mutatni, én hogyan tettem közzé. Mivel egy projekt sosincs kész, ezért egy fejezetben lesznek felsorolva olyan további funkciók és fejlesztési lehetőségek, amikkel jobbá lehet tenni a feladatot. A tesztelés fejezet után pedig összegzem majd, hogy sikerült-e elérnem azt, amit kitűztem célnak magamnak.

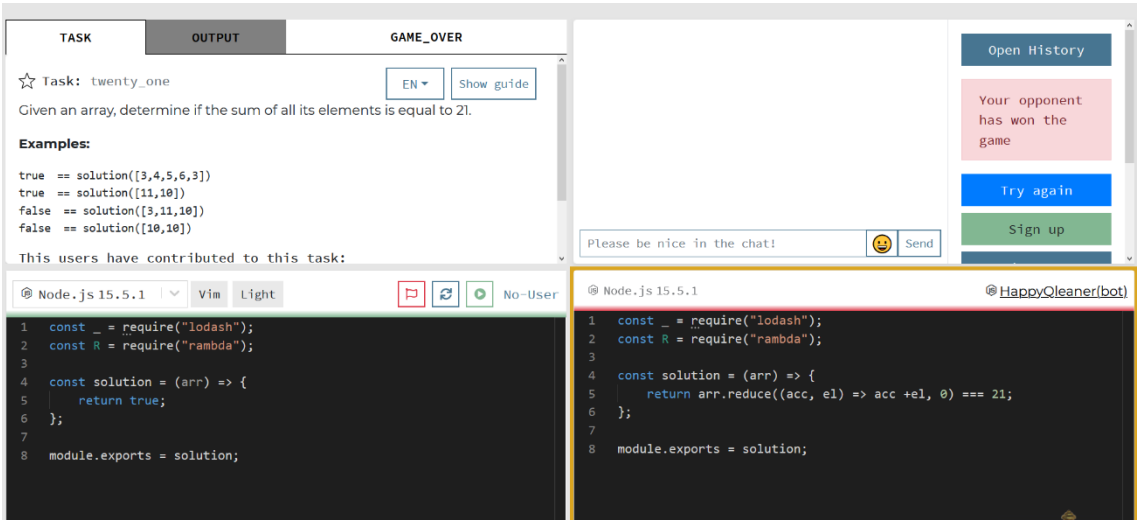
2 Irodalomkutatás

Ebben a fejezetben először bemutatom a szakdolgozat témájához leginkább hasonló weboldalakat, ezekről egy rövid értékelést teszek, hogy a szakdolgozat témájához képest mennyiben más, majd egy rövid személyes értékelést adok róluk. Ezek után kifejtem, hogy egy webalkalmazás általában milyen elemekből épül fel, ezekre fejtek ki pár példát és mondom el röviden, hogy a felhozott rendszerek közül én melyiket fogom használni és miért.

2.1 Hasonló rendszerek

2.1.1 Codebattle

A legjobban a projekthez hasonlítható oldal a Codebattle[1][2]. Az oldalt bejelentkezés nélkül is ki lehet próbálni, amennyiben a weboldalt megnyitva a „TRY SIMPLE BATTLE” gombra kattintunk.

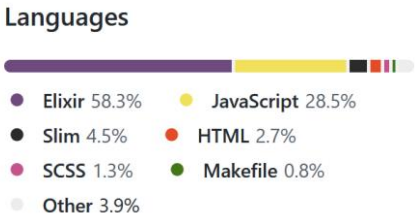


Ábra 2.1Az oldal játszó felülete

Az említett gombot megnyomva kerülünk a 2.1-es ábrán megjelenő oldalra. Ezen láthatjuk a 4 jól elkülöníthető részét az oldalnak. A bal felső sarokban van leírva a megoldandó feladat, jobb felül látunk egy chat ablakot, jobb alul a mi szövegmezőnket, ahova a megoldásunkat tudjuk írni, illetve jobb alsó sarokban az ellenfelünk kódját. Az oldalra regisztrálást követően látjuk a bejelentkezett felhasználóknak szóló főoldalt.

[1] https://codebattle.hexlet.io/
[2] https://github.com/hexlet-codebattle/codebattle

Itt tudunk saját játékot létrehozni, illetve mások által kreált versenyéhez csatlakozni. Van egy külön fül tornáknak, amik több körös játékot foglalnak magukba.



Ábra 2.2 Az alkalmazás nyelvbeli felosztása[1]

A weboldal nagyrésze Elixirben és JavaScriptben van írva, ezeknek a százalékos megosztásáról láthatunk egy kis grafikont a 2.2-es ábrán. A játék működéséhez szükséges szövegdoboz, ahova írhatjuk a kódunkat több elterjedt program nyelvet is ismer. Ezek közül egy pár példa a Node,js, C++, C#, Java, Python, TypeScript, illetve pár kevésbé ismertet, például: Haskell, Elixir. A weboldal a tesztek - amihez a mi kódunk helyességét viszonyítja, hogy jó megoldást adtunk le - lefuttatását a mi böngészőnkbe végzi, nem küldi el egy a weboldal mögött működő szolgáltatásnak. Minden beregisztrált felhasználóhoz van egy „elo_ratirng” nevezetű szám rendelve, ami a játékos tudását szimbolizálja. Ez minden esetben 1200-ról indul a beregisztrálást követően. Vereség esetén lejjebb, győzelem esetén feljebb megy ez a szám.
Ami feltűnően hiányzik erről az oldalról:

- Egy gyakorlás lehetőség, vagyis nem élő ember elleni megmérkőzés. Lehetőség van úgynevezett botok ellen megmérkőzni, viszont ez sem számít gyakorlásnak, mivel itt is van egy ellenfelünk, aki oldja meg a feladatot velünk együtt, és veszíthetünk, ha az végez.
- Olyan saját játék létrehozása, ahova nem tud mindenki belépni ellenünk, hanem kapunk egy linket, amit akár a barátunknak tudunk megosztani, akivel szeretnénk játszani egy kört.
- Egy automatikus rendszer, ami a tudásban hozzánk közel álló ellenfelet sorsol be, ez itt úgy van megoldva, hogy a játék létrehozásnál egy egytől ötig terjedő skálán lehet megadni, hogy mennyire vagyunk tapasztaltak.

Saját vélemény: A weboldalt könnyű használni a leegyszerűsített kinézete miatt, egyszerű értelmezni, hogy melyik gomb mit csinál. Maga a beléptetett főoldalon nincs felesleges menüpont, csak a legfontosabb funkciók vannak ott. Hátrány az, hogy a játékos oldal alján van egy visszajelzést küldő gomb, ami miatt az egész weboldal, ha nem is sokat, de összemegy, ezzel fel nem használt rész keletkezik, amivel akár pár sorral több kódot is láthattunk volna egyszerre. Mivel egy nagyon új oldalnak számít, ezért a lehetőséget megadták, hogy botok ellen is lehessen játszani, ami nagyon sokat segít a kevés aktív játékos szám kiküszöbölésén. Ezeknek a botoknak vannak előre definiált nehézségeik, ezért nem meglepetés, hogy egy mennyire jól „játszó" ellenfelet választunk.

2.1.2 Codewars

A beregisztráláshoz egy nagyon egyszerű feladatot kell megoldani. Az oldal hangsúlya főleg a gyakorlás részen van. Támogatott programozói nyelvek közül a 2.1.1-ben említett weboldallal szemben itt sokkal több van, pontosan 55. Bejelentkezést követően megkapjuk a feladat választó ablakot, illetve egy listát, ahol ki tudjuk választani magunknak azokat a programozói nyelveket, amikkel szeretnénk feladatokat megoldani. Feladvány fajtái közül tudunk választani, hogy az alapelveket szeretnénk elsajátítani, egyre nehezebb feladatokat kapjunk, vagy az oldalra bízzuk és majd ő kisorsol nekünk egy véletlenszerű munkát.

Ami a szakdolgozatban eltervezett feladathoz képest más:

- Az oldal a gyakorlásra teszi a hangsúlyt, más játékos ellen nem lehet megmérkőzni.
- A főoldal feladvány előnézete, illetve kiválasztási része nagyon kicsi helyen terül el, sok a felesleges, ki nem töltött terület

Saját vélemény: A beléptetett főoldal nagyon rosszul van összerakva, mivel az oldal fő funkciója, az, hogy tudjunk gyakorolni, alig kapott helyet. A feladat előnézet szintén kevés helyet kapott, mindössze 6 sor látszódik egyszerre. Az oldal nagyítására nem reagál. 1080p-nél nagyobb monitoron több a kihasználatlan hely, mint ahol van valami. Egy nagy pozitívum, hogy a weboldalon van egy program, amire, ha belépünk, akkor a programban résztvevő cégek állas hirdetéseiről kapunk értesítést, illetve tudunk az első interjúra bejutni.

2.2 Keretrendszerek

Mivel a manapság létrehozott weboldalaknak számtalan feladatuk van, amit végre kell hajtaniuk, rengeteg komponensből állnak, ezért egyre jobban elterjedtek a már előre létrehozott keretrendszerek. A keretrendszerek a már megírt könyvtárakat foglalják össze egy bizonyos célra. Ez a cél lehet Frontend alkalmazások létrehozása, Backenden futó REST API üzemeltetése. 2021-ben a JavaScript nagyon elterjedt, sokféle feladat ellátására használják, legyen az Frontend, Backend. Ennek köszönhetően ma nagyon sok népszerű keretrendszer épül rá, amiből a későbbi fejezetekben pár be is lesz mutatva.

2.2.1 Nem elkülönített Backend és Frontend:

A weboldal felépítése lehetne olyan, hogy minden része a webalkalmazásnak a felhasználó által futtatott böngészőben teljesülne. Erre egy jó példa a HTML, CSS, JavaScript technológiai hármas. Ennek több hátránya is lehet, ezekből egy pár felsorolva

- A szakdolgozat feladata egy olyan funkciót kell megvalósítson, ami nem egy idő alatt teljesülne az összes felhasználó gépén. Ez a saját kódjának a fordítása, a kész programot tesztek ellen futtatása. Ez egy processzort használó művelet, amiből rengeteg fajta van, gyengétől nagyon erősig terjedően, ezért két teljesen eltérő számítógép konfigurációval felhasználó nem azonos eséllyel indulna neki a játéknak.
- Amennyiben az alkalmazás nem elkülöníthető rétegekben épülne fel, ezért akár egy későbbi telefonos alkalmazás fejlesztése esetén teljesen elölről kell kezdeni a fejlesztést és a már működő Backend üzleti logikát nem tudjuk újra felhasználni

2.2.2 Elkülönített Backend és Frontend:

A 2.2.1-es bekezdésben említett hiányosságok kiküszöbölése érdekében manapság akár webalkalmazás, akár számítógépes alkalmazás tervezéséről és fejlesztéséről beszélünk, azok jól elkülöníthető rétegekre vannak bontva. Ezek a részek:

- Backend: olyan alkalmazás, amit a felhasználó nem lát. A programrész célja, hogy minden logikai művelet itt hajtódjon végre. Ennek a futtatása a kód író dolga, nem függ attól, hogy az alkalmazást használó felhasználónak milyen gépe van. Mivel nem a felhasználó webböngészőjében fut, ezért ez a rész több helyen is felhasználható lehet,

például egy web-, telefonos- illetve asztali alkalmazás is használhatja egyszerre.

- Frontend: ennek a célja nem a logikai műveletek elvégzése, hanem a Backend részből kapott adatokat jelenítse meg, az esetleges felhasználó által generált adatokat továbbítsa a Backend felé.

2.2.3 Node.js

A Node.Js egy szerver oldali JavaScript környezet. Ez a Google által létrehozott V8 motorjára épül. Ez hajtja a ma legtöbbet használt webböngészőt, a Google Chrome-ot is. Maga a Node az C és C++ nyelven van implementálva, az alacsony memória használat és a teljesítmény érdekében. Ennek a környezetnek az elterjedésében nagy szerepet játszott, hogy nem szükséges neki, hogy több szálon fusson. Ezt a funkciót el tudja érni úgy, hogy aszinkron esemény modellt használ. A többi nyelvhez képest ez annyiban eltérő itt, hogy amíg máshol ez a funkció egy külön osztálykönyvtár telepítése szükséges, addig a Node.js ezt alapértelmezetten támogatja. Ameddig a Frontend oldalon a JavaScript nagyon elterjedt, addig a Backend oldalon nem annyira volt használva ez a programnyelv, egészen az utóbbi évtizedig. Mára már a szerver oldali JavaScript fejlesztés egyik legtöbbet használt környezetévé nőtte ki magát. Az egyik ilyen nagy elterjedést kiváltó ok az „npm”, vagyis a Node package manager. Ennek köszönhetően egyszerűen telepíthetünk mások által megírt és fenntartott osztálykönyvtárakat, és ezeknek a követelményeit, amire épülnek. Egy másik ilyen lehet az is, hogy a fejlesztőknek nagyon csábító volt az, hogy mint a Frontenden, mint a Backenden ugyanazt a programozási nyelvet használják.[5]

2.2.4 JavaScript és TypeScript

JavaScript alapértelmezetten nem használ típusos változókat. Ez azt jelenti, hogy amikor létrehozunk egy változót, akkor nem kell megmondanunk a kódunkban, hogy éppen egy számot, szöveget vagy bármi mást hoztunk létre. A változók elé elég beírnunk a „let”, „const” vagy a „var” kulcsszavakat. Ennek vannak előnyei és hátrányai. Hogy ezeket jobban megértsük, a JavaScript egyik tulajdonságáról beszélni kell előtte, ami a „Hoisting”. Ennek az a lényege, hogy a nyelv motorja programkódunk futtatása előtt végig megy a változókon és a metódusokon, azokat a programunk tejére helyezi át. Ez a

működés azonban nem minden típusú változóra érvényes. A következő táblázatban látható, hogy melyekre érvényes ez.

Változó	*Bármilyen érték esetén használható*	*Hoistolás*
var	Igen	Igen
let	Igen	Nem
const	Igen	Nem

Táblajegyzék 2.1 – JavaScript változók működésének ismertetése

A táblázatból kiderül, hogy a 3 változó deklarálásához használt kulcsszavak közül csak egyikre érvényes a Hoistolás. Az előnye annak, hogy Hoistolva van a változónk a „var” kulcsszóval:

- Nem kell attól tartanunk, hogy rossz helyen deklaráljuk a változónkat, mert a program fordítása közben a programkódunk tetejére fog kerülni.
- Nem kell azon gondolkozni, hogy milyen típus legyen a változónk, hanem elég csak a „var” kulcsszót megjegyeznünk, és ezt használhatjuk mindenre.
- Mivel a programunk elejére kerül a változónk, ezért globális szinten használhatjuk, nem kell minden metódusban külön létrehoznunk.

Az előnyökkel azonban hátrányok is járnak, amik közül egy pár lehet, hogy:

- Nagyobb kód esetében, ha nem ott deklaráljuk a változókat, ahol használjuk, akkor nagyon átláthatatlan lesz a kódunk.
- Megeshet az, hogy amíg a kód első részében használunk egy változót szám tárolására, addig később ugyanezt a változót felhasználjuk egy szöveg változó tárolására, akkor előfordulhat, hogy később nem tudjuk, melyik mit reprezentál pontosan.

A „var” változóval szemben azonban ott van a „const” és a „let”, amire nem érvényes a Hoistolás. Ezeket a változókat metódusokban, vagyis kód blokkokban hozzuk létre, és csakis abban a blokkban érhetőek el. A „const” és a „let” abban különböznek, hogy amíg a „let” változókat módosíthatjuk később, addig a „const” változóknak csak ott adhatunk meg éréket, ahol létrehozzuk. Ezzel a „var” egyik hátrányát valamennyire kiküszöböltük,

hiszen tudjuk, hogy mire érvényesek a változóink, ezeket később nem tudjuk átírni. Ezeket összesítve azonban a nagyobb terjedelmű kód bázisoknál nagy kihívás lehet az, hogy nem tudjuk pontosan milyen típusú változónk van, egy metódus mivel tér vissza. Ezekre a problémákra ad megoldást a TypeScript, ami a JavaScript kibővítése olyan értelemben, hogy a JavaScript „const" és „let" változóit használják, viszont megadhatunk a változóknak típusokat, ha szeretnénk.

```
1 var costumer = 'Jhon';
2 console.log(`Name of a costumer: ${costumer}.`);
3 costumer = 5;
4 console.log(`Number of costumers: ${costumer}.`);
```

A fenti kódrészlet a JavaScriptben egy a szabályokhoz igazodó és lefutó program. Lehet az, hogy eleinte egy szöveg tárolására használunk egy változót, de később egy szám elmentésére használjuk. A kód lefutása utána konzolunk kimenete a következő lesz:

```
1 "Name of a costumer: Jhon."
2 "Number of costumers: 5."
```

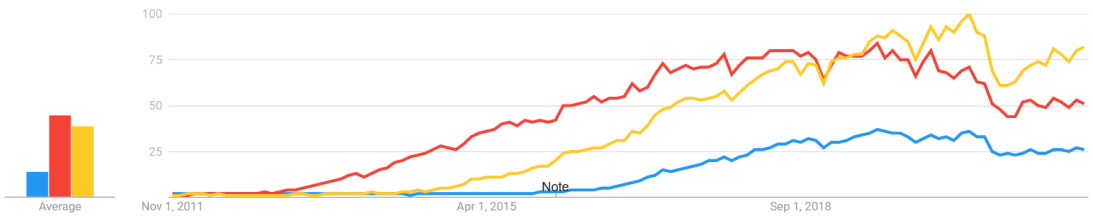
Ugyanezt megpróbálhatjuk TypeScriptben megvalósítani úgy, hogy megadjuk a változónak, milyen értéket tárolunk a következő kódrészlettel:

```
1 let costumer: string = 'Jhon';
2 console.log(`Name of a costumer: ${costumer}.`);
3 costumer = 5;
4 console.log(`Number of costumers: ${costumer}.`);
```

A fejlesztő környezetünk pirossal fogja aláhúzni a kódunk azon részét, ahol más típust akarunk megadni a kezdetinél. A fenti példában itt azt a sort, ahol ötöt szeretnénk megadni. Ez olyan hibát fog dobni, hogy „*Type 'number' is not assignable to type 'string*"", vagyis nem tudunk egy eleinte szöveg tárolásához létrehozott változót később szám tárolására használni. Ennek köszönhetően egy jobban átlátható kód készülhet a TypeScript nyelvben.

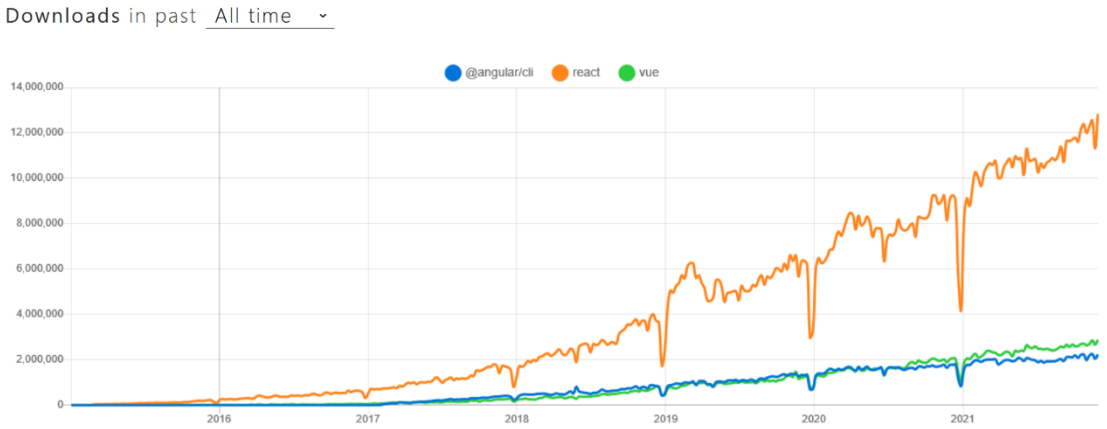
2.3 Frontend keretrendszerek

Ha a 2021-es évben legjobban elterjedt JavaScript vagy TypeScript Frontendre koncentrált keretrendszereket nézzük, akkor három névvel jöhetünk szembe nagy bizonyossággal. Ez a három az Angular, React és a Vue.js. A kijöveteli dátumuk mind 15 és 8 évvel ezelőttre tehető.



Ábra 2.3 A Google Trendsen évek alatt alakult keresési népszerűségük[2]

A 2.3-as ábrán a három említett keretrendszer keresési trendjeit látjuk 2011. 11. 01-től nézve, ahol a sárga a React, a piros az Angular illetve a kék a Vue.js-t reprezentálja.



Ábra 2.4 Összes letöltésük száma[3]

A 2.4-es ábrán látható a három keretrendszer letöltöttsége az évek során a már említett npm rendszeren. Az adatokat az npm trends[3] nezetű oldalon lehet megnézni, ami a hivatalos npm letöltöttségi API adataiból dolgozik.

2.3.1 Angular

A legelső változatán eredetileg 2 Google alkalmazott dolgozott, ami 2008-ban jelent meg és még JavaScriptre épült. A keretrendszernek sikerült felülmúlnia a JQueryt olyan funkciókkal, mint például a kétirányú adatkötés. A kezdeti sikerébe az is nagy szerepet játszott, hogy a keretrendszer fejlesztői egy nagyon részletes dokumentációt publikáltak. 2014-ben bejelentve, majd 2016-ban megjelent a második változata, immáron Angular 2 néven, ami az első változatnak volt az alapokról való újraírása. Hatókör és kontroller

[3] https://www.npmtrends.com/

helyett a hierarchikus elrendezés került főszerepbe. Emellett az újraírással szerepet kapott a TypeScript, hiszen JavaScript helyett ezzel fejlesztették le ezt az újabb változatot. A keretrendszer komponens alapú, ezekkel van felépítve egy Angularral írt program. A Google elkötelezett amellett, hogy minden évben legalább 2 új főverziót kiadnak. Az első változathoz képest az egyik újítás az volt, hogy a szerver-oldali programozást elkezdték támogatni. Ez azt jelenti, hogy az olyan funkciók, mint az adatbázisba írás és olvasás, a szerver fájlrendszerébe mentés és elérés, más szerverekkel való kapcsolat elérhető váltak. Ez egy kliens-oldali programozási nyelvben vagy keretrendszerben nem elérhető teljesen biztonságban, hiszen egy webalkalmazásnál minden a böngészőben futna. Ilyen kliens-oldali nyelvek például a HTML és az AJAX.

2.3.2 React

A React 2013-ban jelent meg. Fejlesztője a Facebook, akik azért hozták létre ezt a nyelvet, hogy a nagyszabású, adat vezérelt programozás kihívásaira és vele jövő problémáira egy megoldást biztosítson. Maga a nyelv a JavaScriptre épül. A fejlesztői annak érdekében, hogy a nyelvet eleinte szkeptikusan vevő felhasználókat meggyőzzék - akik a nyelv sajátos jellemzőitől „féltek” – megírtak egy cikket, aminek a „Why React?” címet adták, hogy meggyőzzék ezeket az embereket. Ezek a sajátos jellemzők később kinőtték magukat olyan szintre, hogy ma már a JavaScript vagy TypeScript keretrendszerek sztenderdjeinek számítanak. Ilyenek például a virtuális DOM-ok, és a JSX. Mint az Angular, egy React alkalmazás is komponensekre épül.[2]

A DOM, vagyis „Document Object Model” a webfejlesztés területén a HTML szerkezetét jelenti, ez adja meg rá a lehetőséget, hogy tartalmi változtatásokat hajtsunk végre a beépített API-k segítségével. Egy HTML DOM fa szerkezetű, amiket könnyen átírhatunk. Ennek az átírásnak a sebessége azonban manapság nagy szerepet kap, mert egy mai átlag weboldal DOM-ja nagy, és ezt szeretnénk sokszor és azonnal átírni. Ezek miatt az szerkesztési sebesség nagyon sok erőforrást vehet igénybe. A virtuális DOM-ok a HTML DOM-ok egyszerűsített és lokális React másolata, ami eltávolodik a böngésző specifikus műveletektől. Mivel ezeknek az írásához nem kell a valódi DOM-ot szerkeszteni, hanem elég a virtuális változatát, ezért gyorsaságban sokkal jobb és nem böngésző specifikus.

A JavaScript XML, röviden JSX, egy szintaxisbővítés a JavaScripthez. Ennek segítségével kiküszöbölhető az a „probléma”, hogy nem tudunk JavaScript és HTML

kódot egy fájlba írni. A JSX-et használva tudunk a DOM-ba tudunk helyezni egységeket a „createElement()” és az „appendChild()” metódusok meghívása nélkül.[4]

2.3.3 Vue.js

Ha a szakdolgozatban bemutatott Frontend keretrendszerek közül a legelső verzió megjelenését vesszük figyelembe, akkor a Vue.js tekinthető a három közül a legújabbnak 2014-es megjelenési dátumával. Kitalálója és fejlesztője Evan You, egykori Google alkalmazott. Az is megemlítésre méltatható, hogy amíg az Angular és React mögött nagy cégek állnak, addig a Vue.js mögött csak Evan van és a keretrendszerhez saját részeket közzétevő közösség. Mint ahogy a két másik keretrendszer, így a Vue is komponenseken alapul. A hivatalos dokumentációja[5] szerint egy progresszív keretrendszer felhasználói felületek fejlesztéséhez. A fő osztálykönyvtára a nézeti rétegre fókuszál, amit könnyen tudunk másik keretrendszerekkel egyszerre használni. Ezek mellet praktikus megoldás lehet SAP-ok, vagyis egy oldalas alkalmazások fejlesztéséhez.[3] A Single Page Application egy olyan alkalmazás forma, ahol sok információ ugyanaz marad, funkciók használatával csak kevés adat kerül frissítésre. Egy példa lehet egy ilyenre a sokak által használ Gmail levelező alkalmazás. A használata közben sok minden ugyanaz marad, mint például az oldalmenü, az oldal tetején található kereső sáv. Egy adat változása esetén csak az a rész kerül újra renderelésre, ahova az az adat tartozik. Egy ilyen művelet tradicionális weboldalakon az egész oldal újra töltését jelentené. Mivel nem kell az egész oldalt újra tölteni és a felhasználó is gyorsabban látja az új adatokat a weboldalon ezért ezzel mind a két fél jól jár.

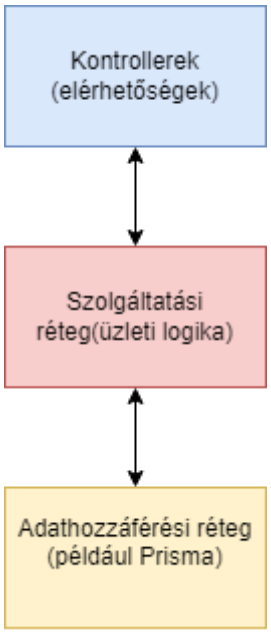
2.4 Backend keretrendszerek

2.4.1 NestJs

A NestJs a Node.js-re épül, annak a már összeválogatott csomagjaihoz adja hozzá a sajátjukat, amit fontosnak tartottak. A keretrendszer alatt a HTTP szervert szolgáló Express fut, ami egy nagyon könnyűsúlyú, gyorsan konfigurálható API építését teszi lehetővé. A keretrendszer bár Node.js-re épül, ám a Nest fejlesztő jobbnak gondolták, ha TypeScriptet használ. A keretrendszer fejlesztőket az architektúra téren az Angular

[4] https://www.w3schools.com/react/react_jsx.asp
[5] https://v3.vuejs.org/guide/introduction.html

inspirálta. Minden külső osztálycsomag telepítési nélkül rengeteg olyan beépített eszközt elérünk, amivel könnyen lehet hibát kezelni, bejövő adatokat ellenőrizni, ellenőrzést hozzáadni a kontrollereinkre és még számtalan mást, ami elengedhetetlen a fejlesztés során. Egy átlagos NestJs alkalmazás 3 rétegre bontható, amit a következő ábra szemléltet



Ábra 2.5 A NestJs 3 rétegű felépítése

Mint a 2.5-ös ábrán is látható, jól elkülönített rétegek vannak meghatározva. A legkülső réteg a kontrollerek, ezek határozzák meg, hogy az alkalmazás az milyen elérési utakon szolgál ki kéréseket. Itt lehet az is megadni, hogy csak már belépett felhasználók érjék el a kontrollereket. A középen lévő réteg szolgál arra, hogy a kontrollereken kapott kéréseket feldolgozza, azok szerint hajtson végre bizonyos logikai műveletek. Az alsó réteg az adathozzáférési réteg, ami biztosítja az adatbázissal való kapcsolatot, ezzel lehet írni és olvasni. Az ábrában példaként említett Prisma egy olyan osztálykönyvtár a NestJs-hez, aminek előre megírt metódusaival lehet írni(*.create()*), olvasni egy(*.findFirst() vagy .findUnique()*) vagy akár több(*.findMany()*) adatsort. Hasonló osztálykönyvtár a C#-ban használt Entity Framework.

2.4.2 Django

Eredetileg Adrian Holovaty és Simon Willison hozta létre a Djangot, amit 2005-ben publikussá tettek nyílt forrás kódú keretrendszerként. Maga a keretrendszer Python

programozói nyelvre épül. Manapság nem csak a Python keretrendszerek közül, hanem általánosságban véve is az egyik legelterjedtebb webes keretrendszer. Mint a NestJs, úgy a Django is hozza magával „out of the box” az olyan fejlesztői eszközöket, amik elengedhetetlenek a mai szoftverfejlesztésnél. Rengeteg féle weboldalt lehet vele fejleszteni, például a Firefox böngészőt fejlesztő Mozilla fejlesztői dokumentációs oldalai is Djangoval készültek, de több másik népszerű oldal is használ valamilyen formában Djangot, mint például az Instagram, Pinterest, National Geographic[6]. A keretrendszer beépített funkcióival segít olyan általános biztonsági támadások kezelésében, mint „SQL injection”, „cross-site scripting” és hasonló. Amiben eltér az eddig bemutatott keretrendszerektől a Django, az az, hogy alapból jön egy adminisztrátori felülettel. Ez automatikusan generálódik attól függően, hogy milyen adat modellek vannak létrehozva a projektben. Egy másik funkció, amit a keretrendszerrel használva igénybe tudunk venni az a gyorsítótár. Ahelyett, hogy az oldalt minden alkalommal újra generálja ugyanazokkal adatokkal, helyette egy gyorsítótárban eltárolja a már kész tartalmat, ezzel felgyorsítva a weboldalak működését.[7]

2.4.3 Laravel

A ma ismert PHP egy 1994-es PHP/FI termék utódja, amit Rasmus Lerdorf készített, ami eleinte CGI (*Common Gateway Interface)* scripteket tartalmazott, hogy nyomon tudja követni oldalának a látogatottságát, aminek a *Personal Home Page Tools*, vagy PHP Tools nevet adta. A legelső PHP webes keretrendszer a Cake PHP publikálta 2000-ben, ami a maga idejében nagy újításnak számított.[8] A PHP-ra épített Laravel keretrendszert Taylor Otwell publikálta 2011-ben. Mind a többi bemutatott keretrendszer, úgy a Laravel is sok olyan alap eszközt hoz magával, ami alapnak számít ma már. A keretrendszer a MVC modellt használja. Az MVC a modell-view-kontroller hármast jelenti. A modell alatt egy olyan osztály értünk, ami az adatbázis tábláival egyenértékű. A view egy olyan osztály, ami a HTML-lel foglalkozik. Bármit, amit egy weboldalon látunk, azt a view részen tudjuk beállítani. Kontroller alatt azt értjük, mint a NestJs esetén. Itt adhatjuk meg

[6] https://www.djangoproject.com/start/overview/ „Sites Using Django” rész felsorolása
[7] https://www.ibm.com/cloud/learn/django-explained
[8] https://www.php.net/manual/en/history.php.php

azokat az elérési utat, ahol az webalkalmazásnak vannak valami elérhetőségei publikálva. Ilyen például egy főoldal, bejelentkezési oldal és hasonlók.[6]

2.5 Backend és Frontend keretrendszer választása és indoklása

NestJs és React TypeScriptben keretrendszereket választottam, mert

- Fontos volt, hogy a Frontend és Backend is ugyanarra a programnyelvre épüljön, ezzel megkönnyítve a munkámat. A választásaim által ezért minden TypeScriptet fog használni.
- Előző fejlesztési tapasztalataim miatt könnyebben fog a projekt elkezdése menni, hiszen nem kell az alapoktól kezdenem a nyelv megismerését.
- Mind a két keretrendszer jól skálázható, ezért, ha a későbbieknek megnövekedne a webalkalmazás látogatottsága, úgy nem kell tartani attól, hogy a keretrendszerek nem bírnák a nagyobb terhelést.
- A React Native segítségével több platformra is elérhetővé tehető az alkalmazás a későbbiekben.
- Mivel mind a két keretrendszer TypeScriptet használ, ezért a felhasználók is ebben a nyelvben fognak programozni, hiszen ennek a kódnak a futtatása a nyelv egyezése miatt könnyen megoldható.

2.6 Websocket

Alap esetben, ha szeretnénk tudni, hogy volt valami adat módosítás az oldalon a betöltés után, akkor egy új kérést kellene küldeni a szerver oldalnak, és ezt bizonyos időközönként - akár másodpercekben értve - újra kellene hajtani. Ez egy pár felhasználós oldalon még kezelhető lenne, azonban amennyivel több felhasználója van egy weboldalnak, annyival több kérés kerül elküldésre a szerver felé. Ez még tovább fokozható azzal, hogy több ilyen komponensünk van, ami szeretné tudni, hogy van-e új adat. Ez a szerver oldalon sok kérés számmal jár, ami lelassíthatja a weboldalunk alap funkcióinak a működését, akár csak egy oldal betöltését. Sok ilyen „van-e új adat” kérés azzal tér vissza, hogy nincs, ezzel pazarolva a szerver erőforrásait. Jobb megoldás lenne, hogy a szerver majd szól a weboldalunknak, hogy van új adat.

Erre találták ki a 2011-ben sztenderdizált WebSocketeket. Egy TCP kapcsolaton keresztül szolgál két irányú kapcsolatot. A HTTP-től független, azonban a HTTP portjaival- 443 és 80 -, kompatibilisnek kell lennie az RFC 6455 értelmében.[7]

A WebSocketek átalános működése, hogy mind a kliens oldal, mind a szerver oldalon implementálunk egy WebSocketet, ugyanazokkal az eseményekkel, a szerver oldalon elérhetővé tesszük egy címen keresztül, amire a kliens oldal fel tud iratkozni, ezzel értesítést kapni, amennyiben egy új esemény hajtódott végre a szerver oldalon, vagy akár a kliens oldal tud adatot küldeni a szerver felé. Amennyiben az egyik fél megszakítja a kapcsolatot, abban az esetben a kommunikációnak vége, egészen addig, ameddig újra nem kapcsolódnak egymáshoz. Sok helyen használatos, az egyik leggyakrabban felhozott példa a chat funkció implementálásához, de több olyan alkalmazás is van, ahol ezeknek a WebSocketeknek a használata ajánlott, például a monitorozó alkalmazások, ahol akár másodpercenként kell lekérni egy adatot, hogy abból egy grafikont tudjon csinálni, vagy akár valós idejű játékokhoz, ahol több játékos játszik egyszerre, és közöttük kell adatot küldeni. 2021-ben a caniuse.com adatai szerint a böngészők 98.36%-a képes erre a kapcsolat forma létesítésére.[9]

A szakdolgozat feladatában a követelmény szerint van egy chat funkció, ahol a felhasználók tudnak kommunikálni. Erre a feladatra jelenthet egy megoldást a WebSocket, ugyanis nem kell másodpercenként újabb kéréseket küldeni a szerver oldal felé, hogy volt-e új üzenet. Elég, ha csak a WebSocketre feliratkozunk, és a kliens oldal kapja az értesítést, ha ilyen esemény történt volna. A chat mellett elvárás az is, hogy a felhasználó kapjon értesítést arról, ha az ellenfél egy bizonyos eseményt hajt végre, és erről az ellenfelét kell értesíteni.

2.6.1 SignalR

A Microsoft által elsősorban ASP.NET környezethez fejlesztett SignalR a WebSocketre épül, azonban más egyéb funkciói is vannak arra az esetre, ha egy kliens nem támogatná ezeket a WebSocketeket, például régi Internet Explorer. Annak ellenére, hogy ASP.NET-re lett fejlesztve elsősorban, több másik nyelvhez is lefejlesztették, hogy így még több platformon legyen elérhető. Az egyik ilyen platform a Node.js, aminek köszönhetően minden Node-ra épülő keretrendszer is tudja implementálni ezt az osztálykönyvtárat.[10]

[9] https://caniuse.com/websockets
[10] https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

2.6.2 Socket.io

A socket.io egy főként JavaScriptben megírt WebSocketet is használó osztálykönyvtár. Több másik nyelvben is elérhető a kliens oldali változata, például: C++, .NET, Rust és még sok egyéb. A SignalRhez hasonlóan a socket.io a WebSocketekre épül, azonban, ha az nem lenne elérhető, akkor HTTP lekérdezésekre vált.[11] Fontos megjegyezni azt, hogy a hivatalos dokumentációjában kiemelik, hogy ugyan WebSocketre épül, mivel extra metaadatokat is használ, ezért egy sima WebSockettel nem lehet rá kapcsolódni, csakis a saját kliensével.[12] Mivel a szakdolgozat feladatában, mind a Frontend, mind a Backend oldalon TypeScript-re épülő keretrendszerek lesznek használva, ezért a feladatban előforduló funkcióihoz, amik WebSocketek igényelnek, a socket.io osztálykönyvtárat fogom használni.

[11] https://socket.io/docs/v4/
[12] https://socket.io/docs/v4/#what-socketio-is-not

3 Követelmény specifikáció

A szakdolgozat biztosítson 3 féle játék fajtát, amik a következőek:

- Gyakorlás: Egy olyan játékmód, ahol nincs ellenfél és nem kell időre megoldani a feladatot.
- Privát játékmód: Két játékos mód, ahol a játék létrehozója kap egy meghívó kódot, és az ellenfél csak az lehet, akinek a birtokába kerül ez a kód.
- Publikus játékmód: A rendszer párosítja össze a két ellenfelet véletlenszerűen. Az nyer, aki előbb oldja meg a rendszer által adott feladatot.

A szakdolgozat feladata legyen elkülönítve 2, jól elhatárolható részre, azaz Backend és Frontendre.

Backend követelményei:

- A 2.4.1-es bekezdésben említett hármas tagolás legyen látható.
- Az alkalmazásnak legyen olyan kontrollerjei, amit bárki el tud érni, mint például a regisztráció és bejelentkezés funkció. Minden olyan végpont, amit a már bejelentkezett felhasználók érhetnek csak el, le kell védeni olyan Guardokkal, amik JWT tokent fogadnak és ezt ellenőrzik.
- Kell lennie egy üzleti logikának, amivel a játékosok által elküldött program kódot lehet fordítani.
- Ezeket a fordított programokat tudja futtatni olyan módon, hogy a kimenetele látható legyen a logikának, hogy már előre megadott helyes teszt eredményekkel tudja összehasonlítani.
- A 2.4.1-es bekezdésben látható ábrán említett Prisma legyen bevezetve, hogy a felhasználói és játék adatokat tudja írni és olvasni amennyiben az szükséges.
- Legyen egy olyan végpont kialakítva, ahova Socket.io-val tud a Frontend alkalmazás csatlakozni.
- Biztosítson egy szám értéket és hozzá egy logikát, amivel könnyen el leget dönteni, hogy a játékos milyen szinten van és a szintjéhez közel álló nehézségű feladatok legyenek kiválasztva.
- Az előző pontban említett pont szerint próbálja meg publikus játékmódban összesorsolni a két ellenfelet, hogy ez a pontkülönbség minél alacsonyabb legyen.

Frontend követelményei:

- Az alkalmazás főoldalán legyen lehetőség könnyen bejelentkezni és regisztrálni.
- A regisztrációt kelljen megerősíteni, ezzel is egy extra védelmet biztosítani a rendszernek.
- Legyen lehetőség jelszőt változtatni, ami egy email cím megadása, és oda egy egyedi kód kiküldése után legyen lehetőség.
- A bejelentkezést követően irányítson a webalkalmazás a játék főoldalára.
- A főoldalon legyen egy játékmód választó, a játékmódhoz szükséges extra funkciók megjelenítése.
- Legyen egy chat ablak, ahol bárki tud üzenetet küldeni, ezeket az üzeneteket bárki megkapja és lássa.
- Látszódjon az öt legutolsó játék részletei, milyen játékmód volt, kivel játszott és ki nyert.
- Egy dicsőségtábla, ahol az 5 legtöbb rang ponttal rendelkező játékos neve legyen megjelenítve a szám értékével együtt.
- A játékos oldala legyen 4 fő részre jól elkülönítve, helyet adva a programozási feladat ismertetésére, egy kommunikációs szövegdoboz kihelyezésére - ahol a játékosok egymással tudnak kommunikálni, a programkód futtatása után a teszt eredmények ismertetésére, illetve egy szövegdoboz, ahova a felhasználók tudják a feladatra megoldásukat gépelni.
- A Frontend kezelje le azt, hogy egy gyakorló játékmódba ne jelenítsen meg az üzenetes szövegdobozt, hiszen nem lenne kivel kommunikálni.
- Ne engedje a játékost új játék létrehozására, amennyiben van egy még futó és véget nem ért játéka.

4 Tervezés

Ebben a fejezetben a feladat felépítését fogom bemutatni különböző ábrák segítségével. Látni lehet, hogy bizonyos események milyen sorrendben következnek, a Backend és a Frontend hogyan kommunikál egymással, illetve a fejlesztés az milyen ütemben fog végződni.

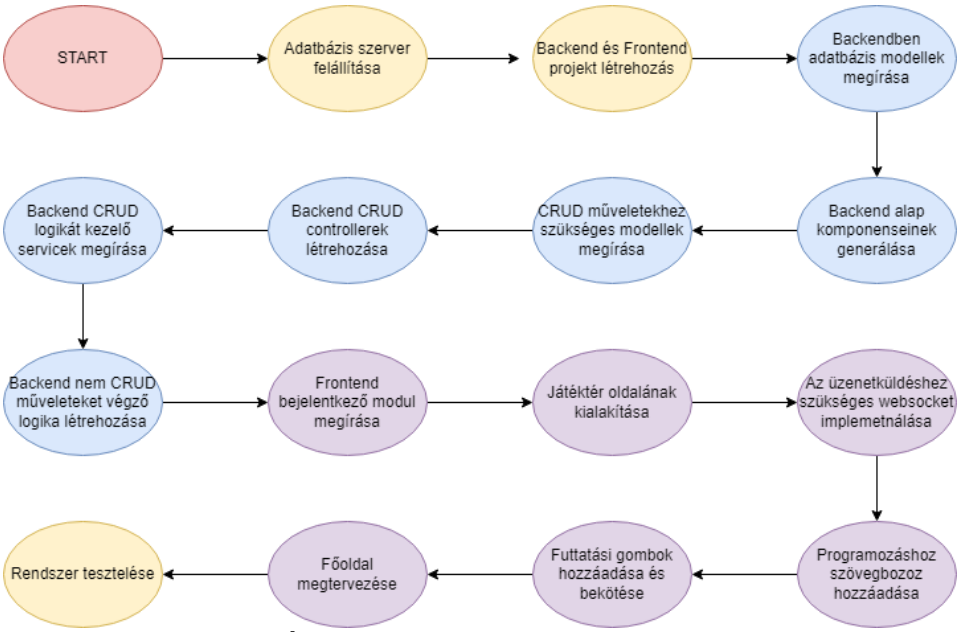
4.1 Adatbázis táblának tervezése

Az adatbázisnak képesnek kell lennie felhasználói adatok tárolására. A játék adatok elmentése mellett szükségünk van modellekre, amik magukat a feladvényokat reprezentálják, ezekhez külön tesztesetek tartoznak. Szükségünk van azt is eltárolni, hogy egy játék kimenetele mi lett, abban kik vettek részt, milyen típusú játékmódról van szó. Minden kód futtatásáról szeretnénk elmenteni adatokat, hogy azokból később akár különböző ranglistákat lehessen összeállítani. A következő ábrákon ezek az adatbázis táblák láthatóak. A CodeBuild táblában lesz tárolva minden futtatás és tesztelés előtt, hogy milyen kódot küldött be a játékos.

Ábra 4.1Az adatbázis tábláinak ábrája

4.2 Fejlesztés tervezett menete

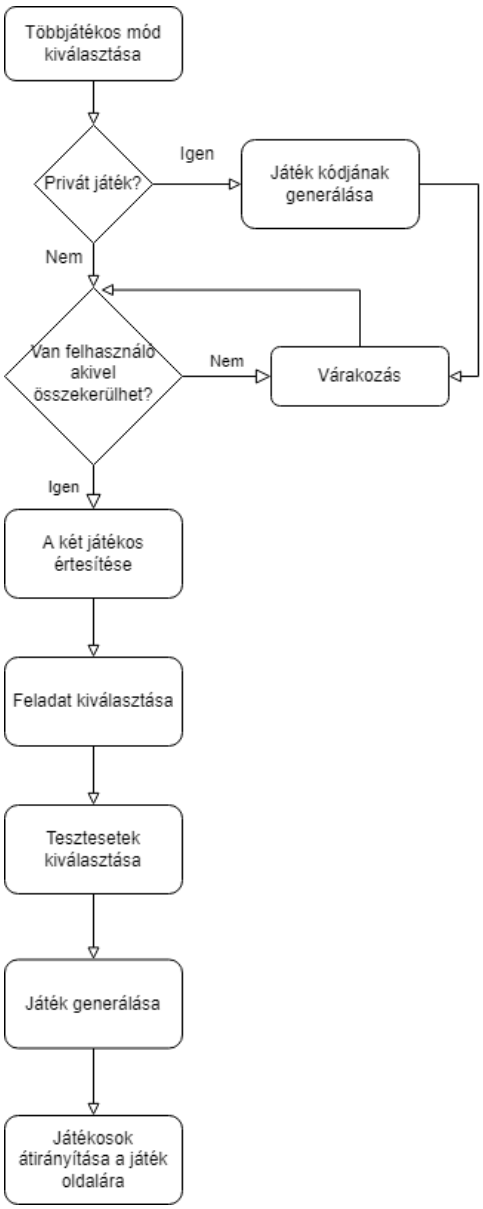
A fejlesztés négy fázisban fog történni. Az első rész, amikor mind az adatbázis szervert elindítjuk, hogy ne kelljen lokálisan adatbázis fájlokat használni, ezzel megkönnyítve azt, ha több gépen folyna a fejlesztés. Ezek után létrehozzuk, mind a Backend mind a Frontend projekteket. Ezek egy Nx „workspace” alatt lesznek, ezzel megkönnyítve azt, hogy mind a két rész hozzáfér ugyanazokhoz a modellekhez A második fázis a Backend alkalmazás lefejlesztése, ami az adatbázisban használt modellek megírásával kezdődik. Ezek után a komponensek generálása következik. Ha ezek megvannak, akkor a kontrollerek fejlesztése kerül sorra, amit az üzleti logika megírása követi. Ebben beleértjük a Prisma használatát, tehát az adatbázissal való összekapcsolást is. Ezek után jön a harmadik rész, ahol a Frontend projekten lesz a hangsúly. A külső oldalaktól kezdve, mint például a bejelentkezés jutunk majd el a belső részekhez, mint például a játékos oldal fejlesztéséhez. Befejezésnek ezeket a modulokat fogom letesztelni.



Ábra 4.2 Fejlesztés menete

4.3 Egy többjátékos mód játék elindítása

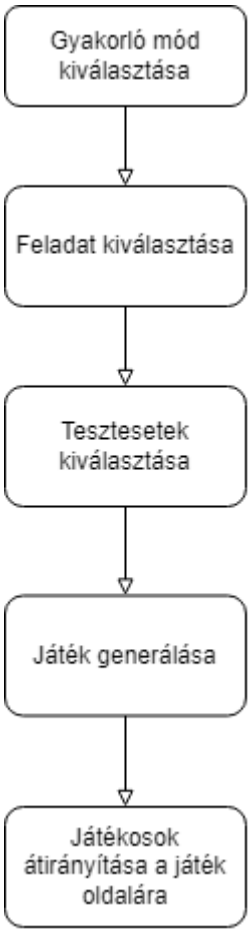
Amennyiben a két többjátékos mód közül szeretnénk egyet használni, először el kell döntenünk, hogy privát vagy publikus játékon szeretnénk reszt venni. Amennyiben privát, úgy egy link kerül generálódásra. Csak olyan felhasználóval kerülhetünk össze, akinek a birtokában van ez a link. Amennyiben publikus meccset szeretnénk, úgy a rendszer addig fog várakoztatni, ameddig egy ellenfelet nem talált. Ha megvan az ellenfelünk, akkor a rendszer kiválaszt egy véletlenszerű feladatot, hozzá választ véletlenszerűen a feladathoz tartozó teszteket, majd az oldal átirányít egy ezekkel az adatokkal ellátott játék felületre.



Ábra 4.3 Többjátékos mód indítása

4.4 Gyakorló játék elindítása

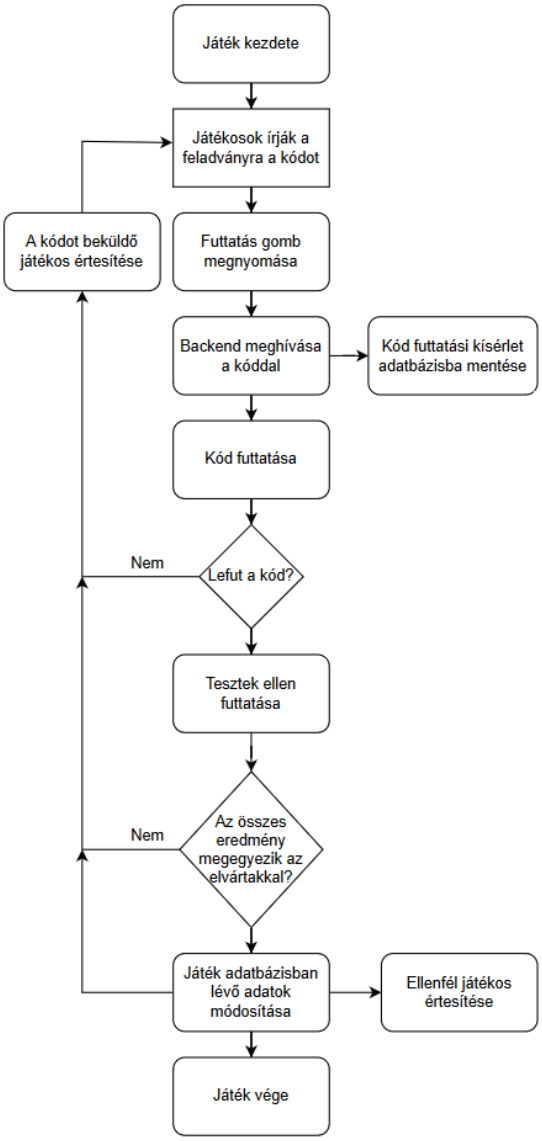
Amennyiben nem szeretnénk valódi ellenfél ellen játszani, úgy tudunk gyakorlás játékmódot választani. Egy ilyen játéknak az elindítása abban különbözik egy többjátékos mód elindításától, hogy nem kell várnunk arra, hogy egy ellenfelet találjon nekünk a rendszer. Fontos megemlíteni, hogy a rendszer választja ki a feladatot, nem pedig mi egy listából.



Ábra 4.4 Gyakorló játék indítása

4.5 Egy játéknak a menete

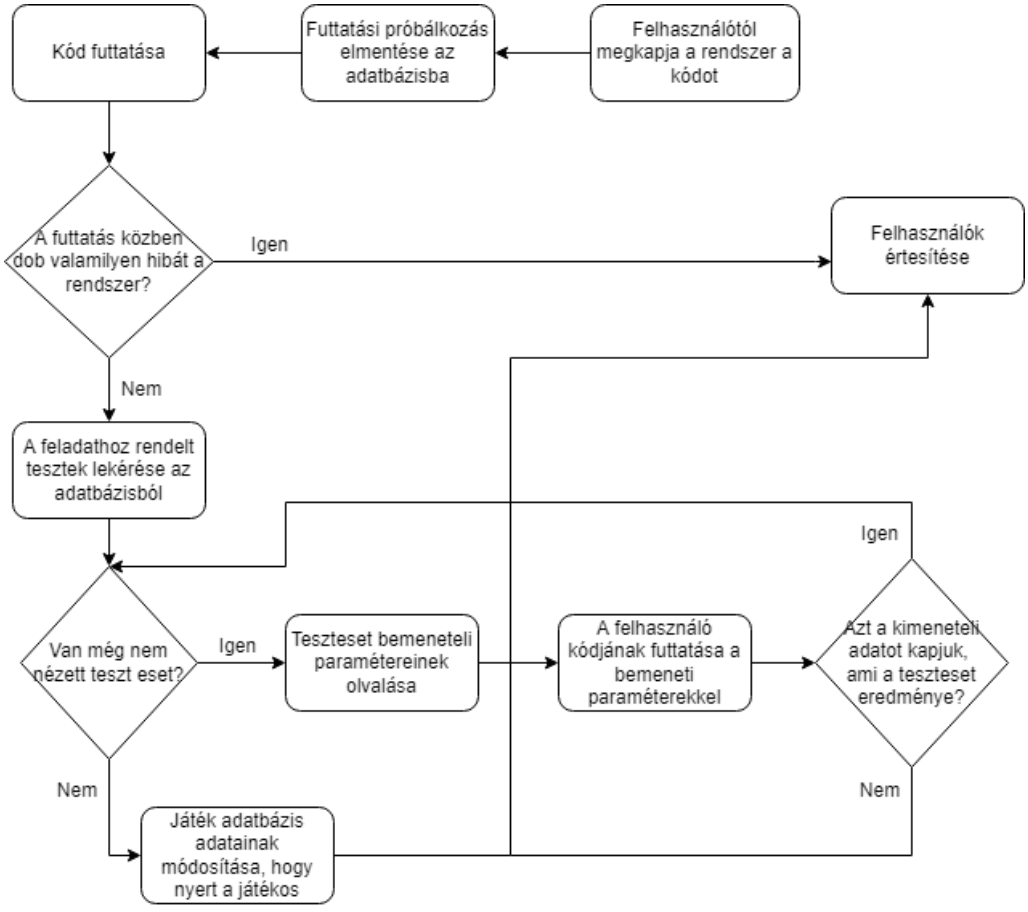
Egy játék a következő a lefutása. A játékos oldal betöltése indítja a játékot. A felhasználók elkezdik írni a megoldó kódot a feladatra, amit a rendszer kisorsolt nekik. Amint úgy érzik, hogy készen van, megnyomják a futtatás gombot. Hogy a két játékos tudja egymásról, hogy a másik megoldást adott le, úgy kapnak egy értesítést. Az elkészült kódot a Frontend elküldi a Backend részére, ami ezt lefuttatja és teszteli, ezek esetleges eredményeikről értesítést küld vissza mind a két felhasználónak. Ez a futtatás és tesztelési folyamat a 4.6-os bekezdésben lesz jobban taglalva. Amennyiben a Backend válasza az lesz, hogy lefutott minden tesztesetre, úgy a kódot leadó játékos nyert, a játéknak vége.



Ábra 4.5 Egy játék menete

4.6 Egy kód futtatása és tesztelése

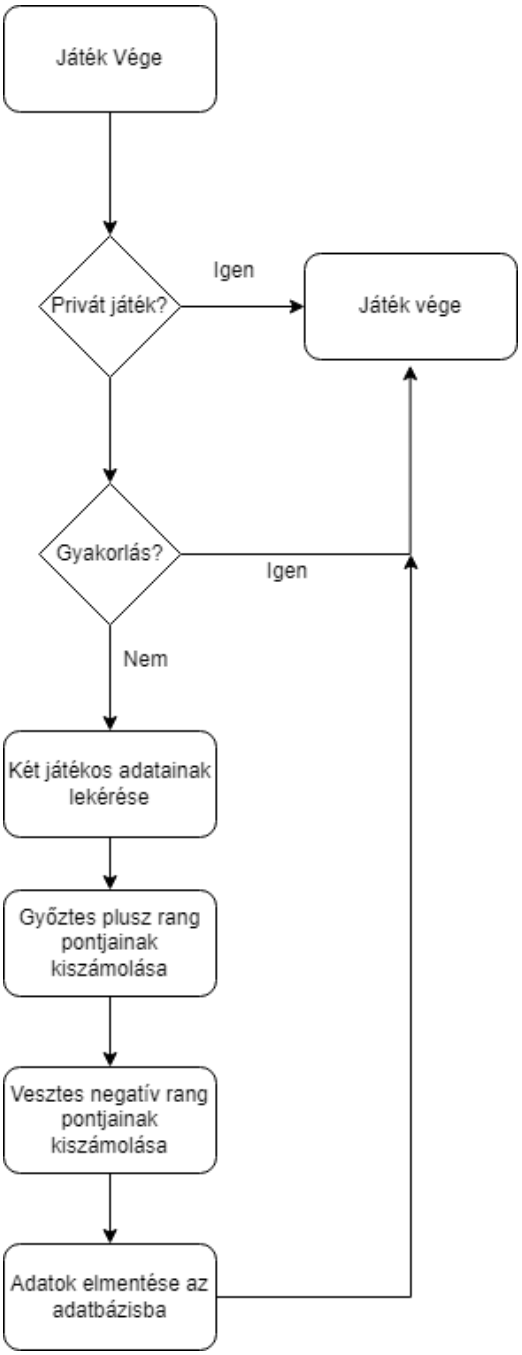
A 4.5-ös bekezdés leírásában és ábráján volt már említve álalánosan, hogy a kódok kiértékelése hogyan történik. Ebben a bekezdésben részletesebben lesz bemutatva. A kódot először simán futtatjuk, hogy van-e benne szintaxis hiba. Amennyiben van, úgy az egész folyamat véget ér azzal, hogy erről a hibáról értesítjük a játékosokat. Amennyiben hiba nélkül lefut, úgy az adatbázisból lekérdezzük, hogy a játékhoz melyik tesztesetek lettek kiválasztva. Ezeken a teszteseteken egyesével végig megyünk úgy, hogy a felhasználó kódjának bemenetére betesszük a teszt bemeneti adatát, újra futtatjuk a kódot és a kód kimenetelét összevetjük a teszt jó kimenetelével. Amennyiben nem egyezik, úgy a teszteset elbukott, ha még lenne hátra teszt, akkor azokat nem futtatjuk. Erről értesítést küldünk a felhasználóknak. Amennyiben a teszten átmenne, úgy ezt addig ismételjük, ameddig nem marad több teszt. Ha nincs, akkor az adatbázisba elmentjük, hogy a kódot beküldő játékos nyert, erről szintén értesítést küldünk a játékosoknak. A játék ezzel véget ért, a Backend elkezdi lefuttatni a játék utáni logikákat.



Ábra 4.6 Kód futtatás és tesztelése

4.7 Játék után lefutó logika

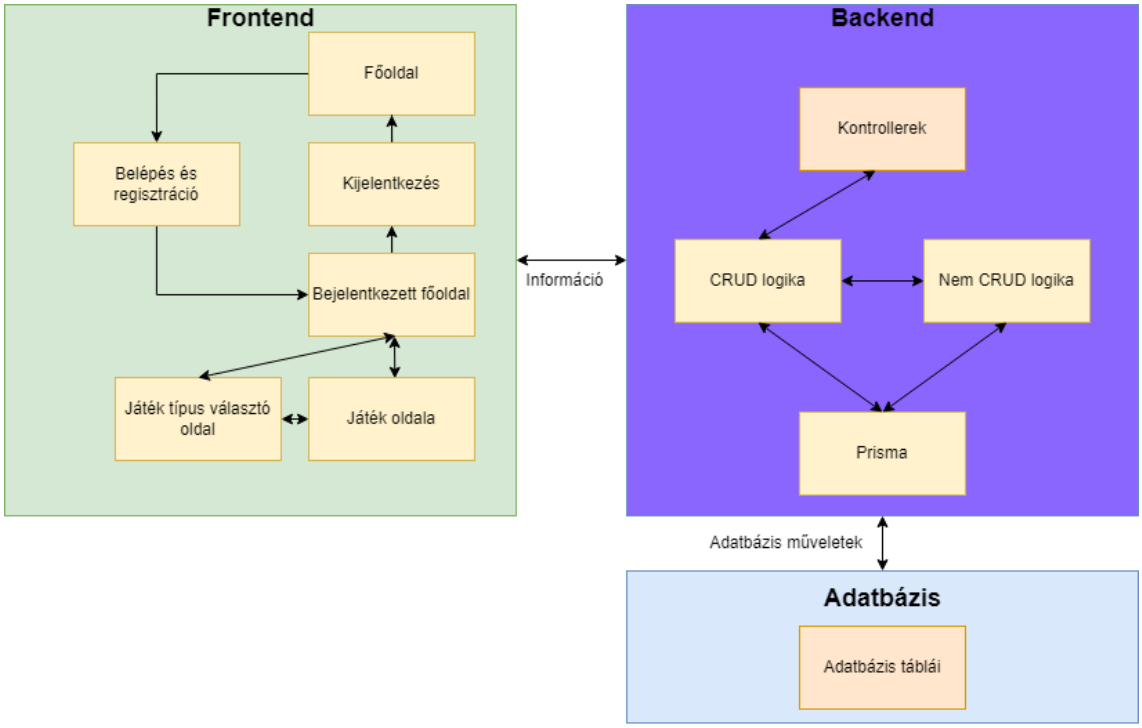
Egy játék vége után a következő logika fut le. Amennyiben egy privát van gyakorlás módja volt a játéknak, úgy semmi extra eseményt nem kell kezelni. Amennyiben egy publikus játékban vett részt a két játékos, úgy a két felhasználó adatait lekérjük az adatbázisból, egy előre meghatározott formula segítségével kiszámoljuk a két játékos rang pontjait aszerint, hogy valaki győzött vagy vesztett.



Ábra 4.7 Játék utáni logika

4.8 Backend, Frontend és az adatbázis kapcsolata

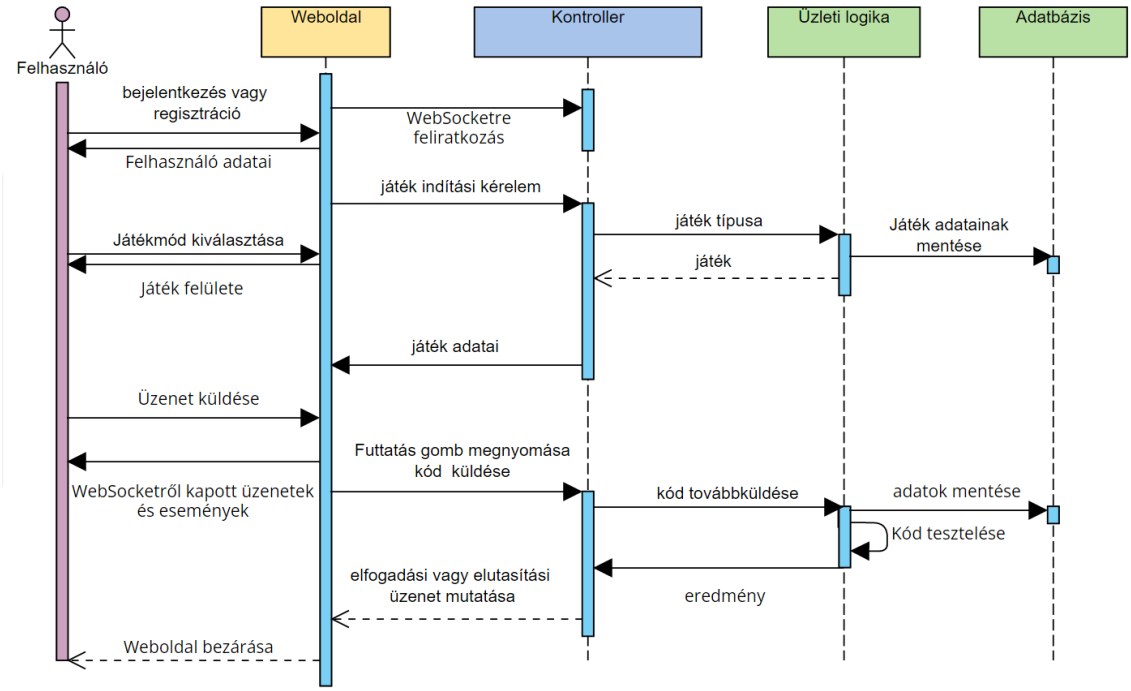
A három réteg, Frontend, Backend és az adatbázis között a Frontend és az adatbázis nem tud egymásról. Kettő között van a Backend, ami feldolgozza a kontrollereken keresztül a beérkezett kéréseket. Amennyiben szükség van itt az adatbázisba írni vagy onnan olvasni, abban az esetben a Prisma segítségével ezt el tudjuk érni. A Backend a Frontend felé már feldolgozott adatot továbbít, így a Frontend feladata nagyrészt ezeknek a megjelenítése. A Backend általában a CRUD logikákat hívja, amennyiben van szükség extra logika meghívására, akkor azokat használja.



Ábra 4.8 Rendszerek kapcsolata

4.9 Egy általános szekvencia diagramm

A következő ábrán egy szekvencia diagramm látható, ami általánosságban véve megmutatja, hogy a felhasználó milyen eseményeket tud előidézni, mint például egy bejelentkezés, játék indítása vagy pedig egy teszt futtatása esetén.



Ábra 4.9 Általános szekvencia diagramm

5 Fejlesztés

Annak érdekében, hogy a fejlesztés és tesztelés fejezetekben a parancsok jobban érthetőek legyenek, ezért a szakdolgozat feladatának adtam egy nevet, ami a CodeItOut lett. Amennyiben előfordul ez a kifejezés, ezalatt egy névre fogok hivatkozni, nem pedig egy bizonyos parancsra, osztálykönyvtárra stb.

5.1 Repository

A feladat elkezdéséhez szükségünk lesz egy repository létrehozására, hogy a git kliensünk tudjon működni. Egy mappát út tudunk kijelölni, hogy a következő parancsot lefuttatjuk egy parancssorban abban a mappában, ahol azt szeretnénk, hogy legyen:

```
git init
```

Mivel fontos az, hogy adatvesztés esetén egy másik helyen is legyen egy replikája a kódbázisunknak, ezért a parancs lefuttatása után a következőt is le kell futtatnunk. `git remote add origin remote.url/project.git`

A 4.2-es terv fejezetben már volt szó az „Nx workspace”-ről, most ezt fogjuk beállítani. Maga az „Nx workspace” egy olyan adatstruktúrát tesz lehetővé, hogy ne kelljen a Backend és Frontend alkalmazásunknak külön repositoryt létrehozni, hanem egy jól átlátható és könnyen tovább bővíthető környezetet biztosít. Az alap fájlok generálásához a következő parancsot kell futtatni:

```
npx create-nx-workspace --preset=ts
```

A „--preset=ts” opcionális kulcsszóval adjuk meg, hogy egy TypeScript alapú struktúrát szeretnénk. A parancs futtatása során kérni fog egy nevet a környezetnek - a szakdolgozat esetében ez a már említett CodeItOut - és hogy szeretnénk-e használni az „Nx Cloud” szolgáltatás. Ez ugyan csak opcionális, viszont ajánlott, ugyanis a használatával a projektek futtatásának az ideje jelentősen lecsökkenhet.

5.1.1 Backend projekt létrehozása

A 2.5-ös fejezetben kiválasztott NestJS backend alkalmazást a létrehozott repositoryba kettő parancs lefuttatásával lehetséges. Első, hogy az feltegyük az Nx NestJs függvény könyvtárát a következő paranccsal:

```
npm install -D @nrwl/nest
```

Miután ez a folyamat lefutott, az alkalmazás generátort kell futtatni, ami a következő:

```
nx g @nrwl/nest:app codeitout-backend
```

5.1.2 Frontend projekt létrehozása

Szintén a 2.5-ös fejezetben kiválasztott Frontend alkalmazás környezetet a már létrehozott Backend mellé szintén két parancs lefuttatása szükséges. A folyamat a Backend létrehozásához hasonló, csak annyiban tér el, hogy a ’nest’-et átírjuk ’react’-ra, azaz:

```
npm install -D @nrwl/react
nx g @nrwl/react:app codeitout-frontend
```

5.1.3 Projektet elindítása és fordítása

A repository és az alkalmazások generálása után még nincs lehetőségünk futtatni a projekteket, ezekhez nekünk a ’package.json’ fájlban, a script objektumba a következő sorokat beleírni:

```
1 . . .
2 "scripts": {
3   "start": "nx serve",
4   "start:backend": "npx nx serve codeitout-backend",
5   "start:frontend": "npx nx serve codeitout-frontend --port 5465",
6   "build": "nx build",
7   "test": "nx test"
8 },
9 . . .
```

5.2 Adatbázis

Ahhoz, hogy azt a követelményt teljesíteni lehessen, miszerint meg kell őriznünk játékos és játék adatokat, szükségünk lesz egy adatbázisra. A projektben 2 féle adatbázis lesz használva, attól függően, hogy a projekt milyen környezetben van futtatva.

- Amennyiben a fejlesztői gépen, abban az esetben:
    - Sqlite: Egy fájl alapú adatbázis, aminek futtatásához nem kell sok rendszer teljesítmény, ezért a gyengébb gépeken is könnyen használható. Még egy előnye, hogy nem kell semmi extra szervert futtatni hozzá, hanem elég a generált adatbázis fájl.

- Amennyiben a már hosztolt változatot szeretnénk futtatni:
    - PostgreSQL: A már lefordított és éles környezeten futó projekthez az adatbázist egy PostgreSQL fogja szolgáltatni. A Sqlite-al szemben a PostgreSQL futtatásához szükséges egy adatbázis szerver, aminek használata már jóval erőforrás igényesebb, mint egy fájl létrehozása. A fokozottabb biztonsága miatt jobb választás éles környezetre.

Ahhoz, hogy ezeket az adatbázisokat elérjük szükségünk van egy olyan függvénykönyvtárra, ami támogatja mind a két típust. Erre a célra van használva a már említett Pirsma. Ennek a használatával egy modell fájl megírása után könnyen tudjuk kezelni az adatbázisunkat, migrálhatunk, ha új mezőt vagy táblát vettünk fel stb.

5.2.1 Prisma telepítés

A Prisma feltelepítéséhez mint ahogy a 5.1-es repository részben, itt is a parancssorban kell futtatni utasításokat.

```
npm install prisma --save-dev
```

A telepítés után tudjuk inicializálni az osztálykönyvtárat a következő képpen:

```
npx prisma init
```

A parancs futtatására 2 fájl keletkezik. Az egyik a ’prisma/schema.prisma’, amiben tudjuk az adatbázis modelleket definiálni és ezeknek a köztes kapcsolatait megadni, illetve a ’.env’, amennyiben még nem létezne.

5.2.2 Prisma konfigurálás

Ebbe az környezeti fájlba létrehoz egy környezeti változót, ami azt mondja meg, hogy hogyan tudunk az adatbázishoz kapcsolódni. Mivel két fajta adatbázis típus lesz használva, ezért attól függően, hogy milyen környezetben van futtatva a projekt kell a környezeti változót beállítani.

Amennyiben fejlesztői környezetben van, úgy az Sqlite elérése a következő sor:

```
1 DATABASE_URL="file:./codeitout.db"
```

Ezzel ellentétben az éles környezetben a PostgreSQL eléréséhez a következő féleképpen kell eljárni:

```
1 DATABASE_URL="postgresql://felhasznalo:jelszo@ip:port/mydb?schema=adatbazis"
```

Ezt a környezeti változót a prisma által generált másik fájlban lehet felhasználni. A ’schema.prisma’ felépítse két részből áll.

- Adatbázis forrása és az osztálykönyvtár generátora.

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}
```

- Az adatbázis modelljeinek felsorolása. Ennek egy példája

```
model Test {
  id         Int    @id @default(autoincrement())
  testNumber Int
  testInput  String
  testResult String
  testType   String
  puzzleId   Int
  puzle      Puzzle @relation(fields: [puzzleId], references: [id])
  games      Game[]
}
```

A modell felépítése kódrészből jól látni, hogy ha xToMany kötést szeretnénk másik táblával, akkor tömbös formában kell megadni, mint típust. xToOne kapcsolat esetén pedig 2 értékre van szükség, 1, ami tárolja a másik tábla idegen kulcsát, és egy típusos változóra, amivel a ’@relation’ attribútummal megadjuk, hogy mit használunk idegen kulcsnak. Ahhoz, hogy tudjuk is használni, kell egy klienst generálni a definiált modellekből a következő parancs futtatásával:

- Ha van változás a modellekben:

```
npx prisma db push
```

- Amennyiben nincs:

```
npx prisma db generate
```

A parancs futtatásával a Prisma létrehoz egy ’PrismaClient’ osztályt, aminek metódusai és változói dinamikusan jöttek létre az adatbázis modellektől függően. Annak érdekében, hogy ne kelljen ennek minden használatnál egy kapcsolatot létrehozni az adatbázissal és

a lekérdezés végeztével szétkapcsolódni, érdemes egy külön osztályt létrehozni, ami ezt a klienst terjeszti ki egy mindig futó kapcsolattal.

5.3 Backend

Ebben a fejezetben a Backend résznek csak a legfontosabb elemeinek lesz egy kis bemutatója, hogy mi és hogyan van használva. A projekt részének fontos eleme, hogy az ne függjön a Frontendtől, ne legyen annak jelentősége, hogy a Frontend az egy weboldal, Android vagy iOS alkalmazás stb. Szintén elengedhetetlen, hogy olyan kód készüljön, ami jól különválasztható modulokra, ezek egy másik projekteknél könnyen felhasználhatóak legyenek újra. Ennek érdekében nem a Backend alkalmazáshoz készítünk saját osztályokat, hanem az „Nx workspace" által nyújtott 'libs' mappába készítjük az osztálykönyvtárakat, ezeket a fő Backend appunkba csak importáljuk.

5.3.1 Library létrehozás és felépítés

Backendes osztálykönyvtáraknál a létrehozás előtt el kell dönteni, hogy ahhoz a könyvtárhoz szeretnénk-e REST API végpontot készíteni. Amennyiben ez nem kell, úgy a következő parancs létrehozza az osztálykönyvtárat és annak struktúráját egy 'service' fájllal és annak a teszt részével.

```
nx g @nrwl/nest:lib my-nest-lib --service
```

Amennyiben tudjuk, hogy kelleni fog elérési pont, úgy a következő paranccsal a service fájlokon kívül létrehoz egy kontroller fájlt és annak az e2e tesztelési fájlját.

```
nx g @nrwl/nest:lib my-nest-lib --service --controller
```

Annak érdekében, hogy az újonnan létrehozott modult lehessen használni a projekthez, a fő applikációnak az 'app.module.ts' fájlban az importok részhez be kell illesztenünk az új modul 'my-nest-lib.module.ts'-ben exportált osztály nevét.

Azért, hogy a Backend és Frontend alkalmazásokban közös modell fájlokkal legyen a fejlesztés, érdemes egy olyan üres szolgáltatót létrehozni, ami nem fog másra szolgálni, csak arra, hogy az interfész és osztály modelleket ide helyezzük, így az importálásuk elérhető a „'import { modell } from '@codeitout/shared/domain'" importokkal. Ezzel a két projekt közötti bármilyen függőséget el tudjuk kerülni.

5.3.2 Hitelesítés és engedélyezés

Biztonság szempontjából fontos, hogy megkülönböztessünk végpontokat, amiket bárki elérhet, és olyanokat, amiket csak a már bejelentkezett felhasználók használhatnak. Hogy ezt elérje a projekt, két különböző részre kell bontani a problémát. Az első rész a hitelesítés, a másik az engedélyezés.

Hitelesítés folyamata annyiból áll, hogy egy nem védett végpontot kell elérhetővé tenni, ahol fogadni lehet azokat az adatokat, amiből el lehet dönteni, hogy melyik felhasználó akarja a rendszert használni. A szakdolgozat feladatában erre a NestJs által biztosított 'JwtModule' osztálya lesz használva, amennyiben a felhasználó által beküldött emailcím és jelszó megtalálható az adatbázisban és megegyezőek az adatok.

A 'JwtModule'-t a fő projektben kell regisztrálni az 'imports' részben, ahol meg kell adni egy kulcsot, hogy mivel titkosítsa a kiadott tokent, és a könnyebbség érdekében globálissá is be lehet állítani. Mivel a szakdolgozat Backend részében nem kell különböző JWT tokeneket kezelni, ezért ez be lesz állítva igazra.

```
1 {
2   ...JwtModule.register({
3     secret: 'someVerySecretSecretForTesting',
4   }),
5   global: true,
6 },
```

Miután ez el lett végezve, a saját servicebe importálása után elérhetővé válik a modulnak több metódusa, mint például a token létrehozása és titkosítása. Miután megnéztük, hogy a végponton érkezett felhasználói adatok léteznek és megegyeznek az adatbázisban lévővel, úgy a következő kódrészlet segítségével tudunk egy tokent létrehozni, amit a végponton keresztül vissza is tudunk adni a kliens oldalnak.

```
1 this.jwtService.sign(payload, { expiresIn: '1d' });
```

Ezt a tokent az engedélyeztetés részben tudjuk felhasználni, hogy csak egy hitelesített felhasználó tudjon egy levédett végpontot elérni. Azért, hogy ezt elérjük, szükséges pár osztályt létrehozni, amik ezeket a tokeneket engedélyezi. Az első, amit létre kell hozni, az egy saját őr osztály, ami kiterjeszti a NestJs 'AuthGuard'-ot. Az egész osztály négy sorból áll, ami annyit csinál, hogy a definiálja a tokennek a stratégia nevét, ami a szakdolgozat projektjében 'jwt' lesz.

```
1 import { Injectable } from '@nestjs/common';
2 import { AuthGuard } from '@nestjs/passport';
3 @Injectable()
4 export class JwtAuthGuard extends AuthGuard('jwt') {}
```

Miután az megvan, szükségünk lesz definiálni egy pontosabb stratégiát is. Ez a 'PassportStrategy' osztály kiterjesztésével érhető el. Ebben az osztályban, a konstruktor ősosztálynak megadunk három értéket egy objektumban, amik a következők:

```
1 jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
2 ignoreExpiration: false,
3 secretOrKey: 'someVerySecretSecretForTesting',
```

Az első a 'passport-jwt' osztálykönyvtárból származik, ami megadja, hogy a lekérdezésből honnan szedje ki a tokent, a második, hogy a lejárt tokeneket ne fogadja el, a harmadik pedig, hogy mi a JWT tokennek a titkos kulcsa. Amennyiben ezt elmentettük, úgy ezt be tudjuk adna abba a modulnak a '.module.ts' 'provide: ' részébe, ahol a tokeneket generáljuk. Mivel ez a modul használva lesz a fő Backend projektben, ezért minden másik modulban is használható lesz, nem kell a '*.module.ts' fájlokban importálni. Engedélyezésnél pedig elég csak a kontrollereket tartalmazó fájlokba behúzni. Miután ezek el lettek végezve, úgy a végpontok biztosítása annyiból áll, hogy egy attribútumként megadjuk az őr osztályunk nevét a következő formában:

```
1 @UseGuards(JwtAuthGuard)
```

A projekt következő futtatása után a végpontot már csak egy olyan JWT tokennel lehet elérni, ami még nem járt le és a hitelesítés szakaszban volt létrehozva.

5.3.3 Socket

A Backend és Frontend közti kommunikáció a REST API végpontokon kívül a Socket.io osztálykönyvtár segítségével történik. Pár funkció a szakdolgozat feladatában, ami ennek az osztály által biztosított kommunikációs csatornáját használja:

- Szöveges üzenetdoboz a játékosok közti valós idejű kommunikációjának érdekében.
- Amennyiben többjátékos játékmódot választ a felhasználó
    - A várólistába be és kilépés
    - Privát játék esetén privát várószobába belépés
    - Játék generálása után a játék oldalra irányításához szükséges adat elküldése

- A játék oldalon amennyiben az egyik felhasználó helyes megoldást küld, úgy a játék végének jelzése

A Backend oldalon szükséges a Socket.io-nak a szerver oldalát létrehozni, míg a Frontend részen ennek egy kliensét.

Backend oldalon egy új szolgáltatás létrehozása után az osztálynak meg kell adni egy attribútumot - '@WebSocketGetaway()' -, ami definiálja azt, hogy az osztály egy websocket elérési pont. Az alapértelmezett portja megegyezik a Backendével, ám ha szükséges, akkor a 'port' kulcsszóval meg lehet változtatni. Az osztályban definiálni kell egy változót, ami a websocket szerverét fogja biztosítani.

```
@WebSocketServer() private readonly wss: Server;
```

Mivel szükséges a kliens oldalról érkező kéréseket lekezelni, ezért az osztálynak létre kell hozni metódusokat, amiknek az attribútum részéhez meg lehet adni, hogy melyik csatornát figyelje. Erre egy példa a következő kódrészlet, ami a globális elkapásáért, és annak továbbításáért felelős:

```
@SubscribeMessage('globalMessage')
handleGlobalMessage(
  client: Socket,
  message: { sender: string; message: string }
) {
  this.wss.emit('globalMessage', message);
}
```

Úgy lehet a többi websocket kliensnek ezt az üzenetet tovább küldeni, hogy a 'wss' változónak meg kell hívni az '.emit()' metódusát, ahol megadjuk, hogy melyik csatornára van küldve milyen üzenet. A példában továbbá az is látható, hogy ez az üzenet lehet akár egy objektum is. Kliens oldalon ez az üzenetküldés lekezelhető a következő kódrészlettel:

```
socket.on('globalMessage', ({ sender, message }) => {
  . . .
});
```

A szakdolgozat feladata során szükség van arra, hogy csak bizonyos kliensek kapják meg az üzenetet, például egy többjátékos módban küldött üzenetről. Erre is van funkció, amivel ezt el lehet érni. A lényege, hogy bizonyos websocket klienseket beléptetjük egy szobába, és a szerverrel csak abba a szóbába küldjük el az üzeneteket. Szobába léptetés szemléltetésére szolgáló kódrészlet:

```
joinClientToRoom(client: Socket, room: string) {
  client.join(room);
}
```

Üzenetet küldeni pedig a következő kódrészlettel lehet szerver oldalról:

```
emitToChannelMessage(
  message: { message: string; sender: string },
  targetChannel: string
) {
  this.wss.to(targetChannel).emit(targetChannel, message);
}
```

5.3.4 Játék generálása és elindítása

Mivel többféle játékmód van, többféle játékos számmal, ezért meg kell különböztetni azoknak az elindulásának a menetét.

A gyakorló módban a többi játékmódhoz képest nem kell WebSocket kapcsolat, ugyanis a gomb megnyomása után már egyből generálhatjuk a játékosnak a játékot. Ezt egy erre a játékmódra létrehozott végpont meghívásával lehet elérni, ami a válaszban visszaadja a generált játéknak az indexét. Ezzel az indexel a Frontend átirányítja a játék oldalára.

Privát játék létrehozásakor szükséges a játékost egy WebSocket szobába beléptetni, hiszen ezzel a szobával lehet majd elérni azt, hogy csakis azok a WebSocket kliensek – ami a mostani esetben csak egy – kapják meg a játék indexét, akiknek szól. Ennek az akciónak a végén az első játékos megkapja a privát játéknak a kódját. Miután ezt elküldte az ellenfelének, ő ezt a megadott helyre beírja és sikeres beküldése után történik a játék generálása. Itt a két játékos ranglista pontját figyelembe veszi, hogy a két játékos tudásszintjéhez közeli feladatot kapjanak. Miután a játék legenerálódott, úgy a WebSocket segítségével elküldi mind a két játékosnak, ők átirányítódnak a játék oldalára, majd kezdetét veszi a verseny.

Amennyiben várólistás játékmóddal szeretne a felhasználó játszani, úgy a várólistás játékmódnak a szintjének – normál vagy rangsorolt – kiválasztása után egy WebSocket kérést küld. A szerver oldal ezt a kérést feldolgozza, és a játékmód szerinti várólistába becsatlakoztatja. Minden 10. másodpercben a Backend végrehajtja a logikát, aminek köszönhetően azonos játékmódra csatlakozott játékosokat összesorolja, kiválasztja a feladatot a két játékos szintjéhez viszonyítva, majd a játék generálása után WebSocketen

keresztül jelez a felhasználóknak, amit a Frontend automatikusan feldolgoz és átirányítja a játék oldalára.

5.3.5 Játékos kódjának futtatása

Biztonság szempontjából a szakdolgozat feladata nagy veszélyt rejt, hiszen ismeretlen kódot kell futtatni. Ezekben a kódokban lehet olyan rosszindulatú rész, ami a webalkalmazást akarná feltörni, hogy kártékony műveletek hajtson végre. Ennek tudatában egy olyan megoldást kell találni, ami ezek ellen védve van. A feladatban erre a vm2[13] osztálykönyvtár lesz használva. A Github[14] oldalukon külön rész van arra szentelve, hogy mindenféle támadás ellen tesztelik a modult.

Ez a könyvtár többféle lehetőséget biztosít egy olyan elkülönítve futott környezet létrehozására, ahol biztonságban lehet kódot futtatni:

- Csak a JavaScript futtatására készült VM modul, ahol csak a beépített objektum függvények elérhetőek.
- A NodeVM modul, ami lehetőséget ad arra, hogy egy olyan környezetben lehessen futtatni kódot, amiben csak a mi általunk megadott modulok vannak, mi állíthatjuk be a környezeti változókat stb. A szakdolgozat feladatához ez lesz használva.
- Egy már buildelt kód futtatására szolgáló rész, a VMScript modul.

Mivel a NodeVM lehetőséget biztosít arra, hogy az elkülönített rendszernek a környezeti változóit a fejlesztő állítsa be, ezért a szakdolgozat feladatához tökéletes. Ennek segítségével könnyen lehet elérni azt, hogy a felhasználó által beküldött kódot több teszt ellen lehessen futtatni. Amire szükség van, hogy a felhasználónak a kódjában legyen benne egy olyan rész, ami kiolvassa a környezetei változóból az éppen aktuális teszt bemenetét. Frontend oldalon ez a rész automatikusan hozzá van rakva a játék kezdetekor, így a felhasználó tudni fogja, hogy honnan jön a bemenet.

Mivel öt különböző teszt van egy játékhoz kiválasztva az összes hozzá tartozóból, így a bemenet és az elvárt kimenet is különbözik. Mint a bemeneti érték megadására, úgy a kimenet olvasása is könnyen megoldható a használt modullal. Amikor egy új NodeVM

[13] https://www.npmjs.com/package/vm2
[14] https://github.com/patriksimek/vm2/tree/master/test

változót hozunk létre, akkor egy paraméterrel meg lehet adni, hogy a ’console.log’ kimenetét tudja az eredeti gép olvasni. A következő kódrészlet szemlélteti egy ilyen NodeVM osztály létrehozását:

```
1 const vm = new NodeVM({
2   timeout: 2000,
3   console: 'redirect',
4   require: {
5     external: true,
6     root: './',
7   },
8   env: {
9     INPUT: test.testInput,
10  },
11});
```

A harmadik sorban lévő kód az, amivel ezt meg tudjuk adni. A kilencedik sorban pedig azt tudjuk megadni, az ’INPUT’ környezeti változó értéke mi legyen, ami tesztekként eltérő.

Miután elindítottuk ezt a külön környezetet a felhasználó kódjával, amennyiben volt ’console.log’ esemény, úgy az ott megkapott érték össze van hasonlítva az éppen aktuális teszt elvárt kimenetével. Mivel egy játékhoz öt teszt tartozik, így ötször van futtatva másik bemeneti környezeti változóval. Amennyiben mind az öt megegyezik, úgy a játékos által megadott kód helyes.

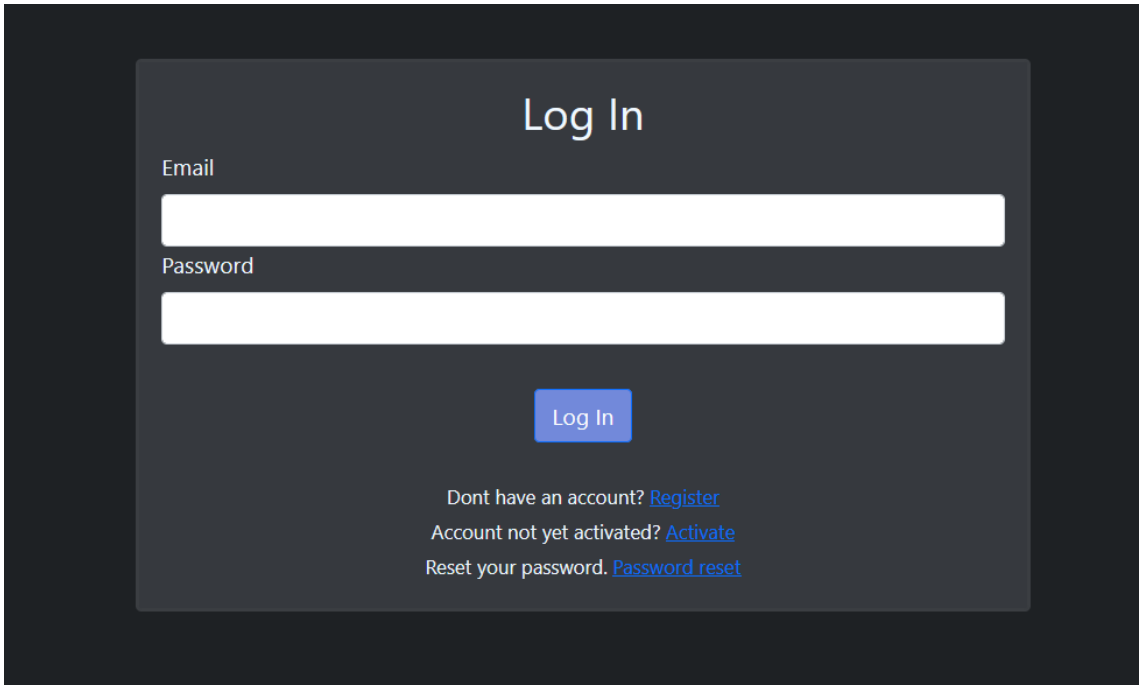
5.4 Frontend

A Backend bemutatása után ebben a fejezetben a Frontendről lesz szó. Ez a rész is több kisebb elemből fog felépülni, itt ezeket komponenseknek nevezik.

5.4.1 Belépés és regisztráció

A weboldal első megnyitása után a belépési oldal fog először szembe jönni. Ahhoz, hogy ez megtörténjen, szükségünk van Frontend oldalról is olyan aloldalakat létrehozni, amik csak bejelentkezés után érhetőek el. Mivel a főoldalnak az elérhetősége - ahol a játékmódot lehet választani - a root domain, ezért onnan át kell irányítani a felhasználókat a bejelentkezési oldalra, ameddig nem rendelkeznek egy megfelelő tokennel. Itt egy

egyszerű űrlap kitöltésére van lehetőség. Ennek az oldanak a kinézete a következő:



Ábra 5.1 Bejelentkező oldal

A bejelentkezésen kívül a felhasználónak van lehetősége:

- A rendszerbe regisztrálni, amennyiben még nem rendelkezik egy felhasználóval.
- Az emailcímet megerősítő kód újra kiküldését kérni emailben.
- Az emailcíménének megerősítése
- Új jelszó megadására, amennyiben megadja a felhasználójának az emailcímet, és az oda küldött kód mellett megadja az új jelszavát.

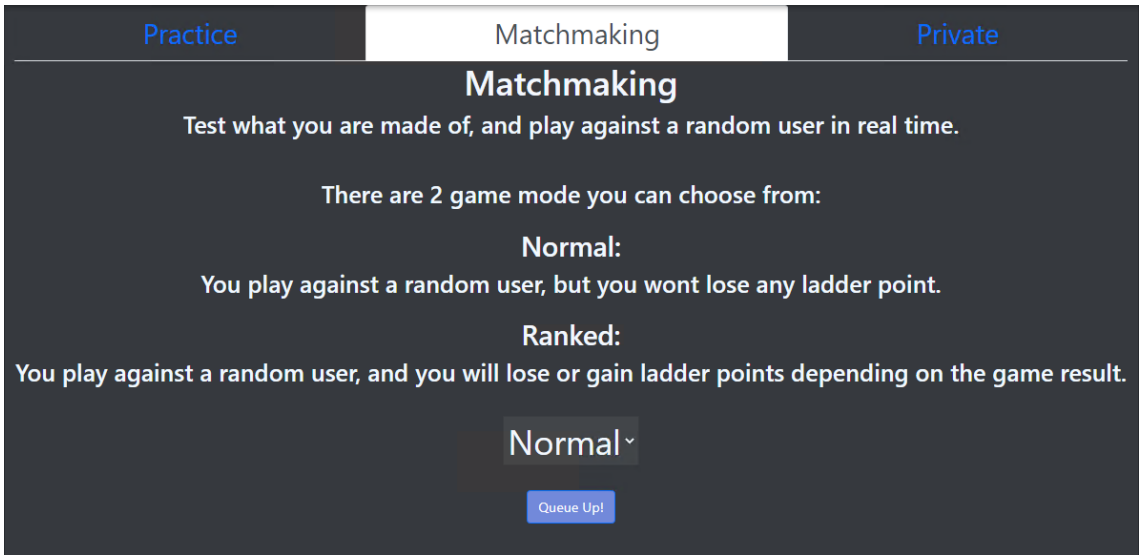
Amennyiben hibás belépési adatokat ad meg, úgy a bejelentkezés egy hibát fog kiírni. Egy sikeres bejelentkezést követően a Backend válaszában lesz egy JWT token, amit a Frontend elment a ’localStorage’-ba. A Frontend oldalon definiálni kell egy modult, ami eldönti azt, hogy a felhasználónak van-e érvénes tokenje. Fontos részlete ennek az ellenőrzésnek, hogy amennyiben van tokenje, az lejárt-e már. Ha lejárt, akkor azt érvénytelennek kell tekinteni. Ezt a következő kódrészt tartalmazó ’authRoute.tsx’ dönti el.

```
1 const token = localStorage.getItem('jwt');
2 if (token === null || token.length < 1) {
3   localStorage.removeItem('jwt');
4   return <Redirect to="/login" />;
5 }
6 const decodedToken: JwtTokenObject = jwt(token);
7 if (new Date().getTime() / 1000 > decodedToken.exp) {
8   localStorage.removeItem('jwt');
9   return <Redirect to="/login" />;
10 }
11 return <div>{children}</div>;
```

Amennyiben a token lejárati ideje az ellenőrzéskor aktuális időnél kisebb, úgy az már lejárt, azt ki kell törölni és a bejelentkezési oldalra kell átirányítani a felhasználót. Amennyiben egy aloldalt csak a bejelentkezett felhasználóknak szánunk, úgy a betöltés előtt le lehet futtatni ezt a logikát.

5.4.2 Főoldal

A bejelentkezést követően a felhasználó átkerül a főoldalra. Ezen az oldalon van a játékosnak lehetősége arra, hogy játékmódot válasszon. Az oldal kinézete a következő, ahol csak a játékmód választó rész látható:

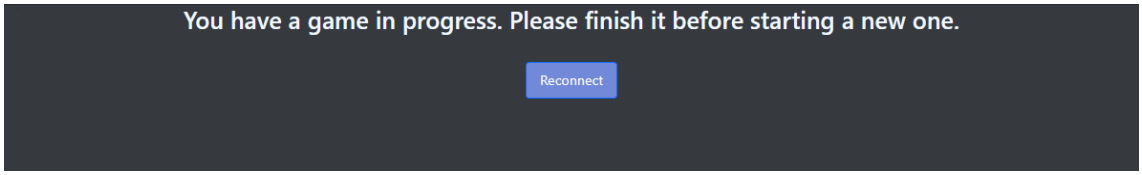


Ábra 5.2 Játékválasztó része a főoldalnak

A felhasználó könnyen tud játékmódot választani, hiszen három oldal található csak, azoknak is csak egy rövid de tömör leírása van. Ezen az oldalon található még egy chat ablak, ahol minden éppen a főoldalon tartózkodó játékossal tud kommunikálni, láthatja

az utolsó 5 játékának a részleteit és egy dicsőségtábla, ami mutatja az összes játékos közül azt az ötöt, akiknek a legtöbb ranglista pontjuk van.

Amennyiben a játékos elindít egy játékot és onnan kilép, fontos elérni azt, hogy addig ne tudjon egy új játékot elindítani, ameddig még van egy befejezetlen versenye. Ennek a funkciónak éppen életbelépett oldala a következőképpen néz ki:

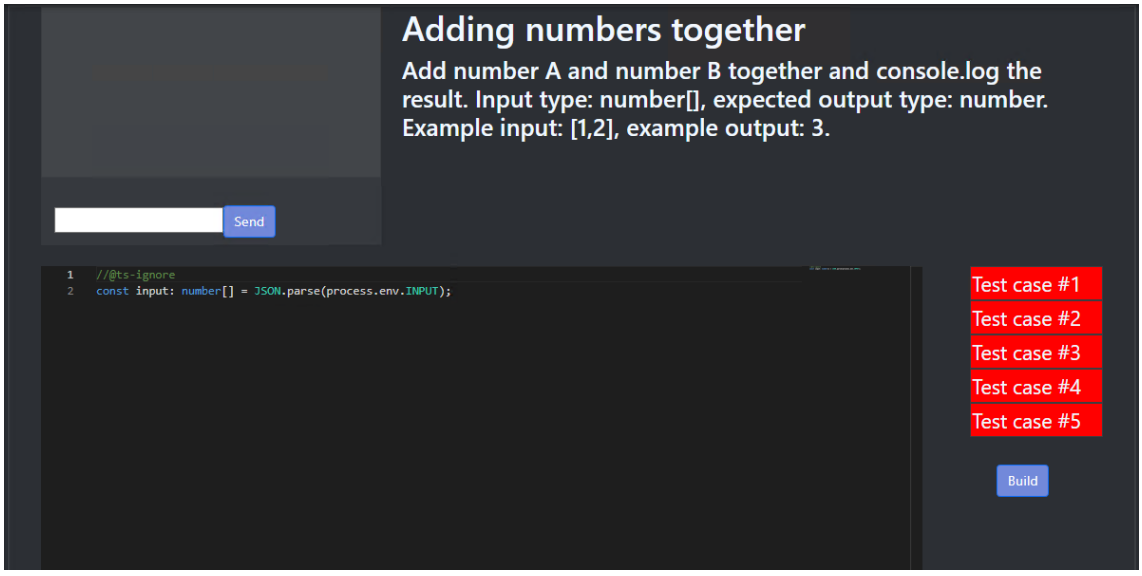


Ábra 5.3 Futó játékra figyelmeztető mező

Fontos megjegyezni, hogy a játékmód választó rész eltűnt, és csak a fenti szöveg látható. A ’Reconnect’ gomb megnyomására visszakerül a még futó játékába.

5.4.3 Verseny oldal

A Frontendnek a másik oldala, amit csak bejelentkezés után lehet elérni, az maga a verseny helyszínét biztosító oldal. Ez attól függően változhat, hogy éppen gyakorlás játékmód van futtatva, vagy egy többjátékos mód. Amennyiben az utóbbi, úgy a következő formában van az oldal megjelenítve:



Ábra 5.4 Egy többjátékos módban elindított játék verseny oldala

Bal felül található egy chat ablak valójában egy rendezetlen lista. Az üzenetek egy tömbben vannak tárolva egy ’state’-ben, ami bővül, ha a socket kap egy új üzenetet. Az üzenetek megjelenítése a következő módon lehet elérni:

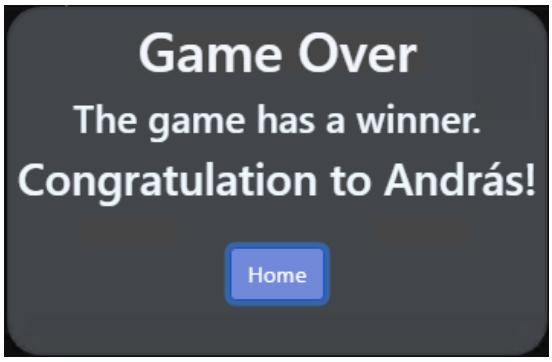
```
1 const renderChat = () => {
2   return chat.map(({ sender, message }, index) => (
3     <li key={index}>
4       {sender}: {message}
5     </li>
6   ));
7 };
```

Ezt a metódust pedig a HTML kód résznél tudjuk felhasználni úgy, hogy meghívjuk ezt a függvényt:

```
1 <ul style={textBoxStyle}>{renderChat()}</ul>
```

Jobb oldalt felül található a feladat, amit a játékosoknak kell megoldaniuk. Ha éppen egy gyakorlás lenne a kiválasztott játékmód, úgy annyiban tér el az oldal kinézete, hogy a felső sorban csak a feladat szövege található. Az éppen aktuális feladat megoldását a bal oldalt alul található szöveges ablakba kell beírniuk. Mivel a szöveges doboz a Microsoft által fejlesztett Monaco Editor[15] komponensét használja, így minden olyan funkciót biztosít, amit egy rendes fejlesztő környezet szövegdoboza. Ez alatt van érte a szintakszis ellenőrzése, illetve éppen aktuális extra metódusok és funkciók ajánlását, ami éppen aktuális a kód kontextusában. Amennyiben a felhasználó úgy érzi, hogy az általa elkészített kód megoldás a feladatra, úgy a ’Build’ gomb megnyomásával azt el tudja küldeni a Backendnek. Ha vannak helyes megoldások, de van legalább 1, ami nem a jó teszt eredményt adta, úgy a helyes tesztesetek szövege kizöldül, a helyteleneké marad piros. A csalás elkerülése érdekében a pontos elvárt teszt eredményt nem jeleníti meg a Frontend, csakis a tesztnek a számát mutatja. Amennyiben a megoldás teljes mértékben helyes, úgy a következő felugró ablak fog megjelenni:

[15] https://microsoft.github.io/monaco-editor/



Ábra 5.5 A játék végeztére felugró ablak

A ’Home’ gomb megnyomásával visszakerül a játékos a főoldalra. Amennyiben többjátékos mód fut, úgy a játékban résztvevő összes játékos megkapja ezt a felugró ablakot, amivel egyidőben kikapcsolja a ’Build’ gomb működését, hiszen annak már nincs jelentősége.

5.5 Fejlesztés közben előfordult hibák és ezeknek a megoldásai

Egy nagyobb méretű projekt esetében mindig előfordulnak olyan hibák, amikre valamilyen megoldást kell találni. A szakdolgozat fejlesztése során is előkerültek ilyenek. Pár példa:

2. Eleinte a terv az volt, hogy a felhasználók Pythonban tudnak feladatokat megoldani, hiszen annak a programnyelvnek könnyű a szintaktikája, nem kell kapcsos zárójelekkel foglalkozni. Mivel a NodeVM-ben meg lehet adni külső osztálykönyvtárat, amit használhat, úgy egy Python kód futtatására alkalmas npm csomag kiválasztása után ez meg lett adva. Az éppen aktuális teszt bemeneti paraméterét a NodeVM-nek a környezeti változóihoz szokásos módon kellett megadni. Viszont ennek a futtatása közben előjött egy olyan kritikus hiba, ami megakadályozta azt, hogy a játékosok Python kódot tudjanak írni. A probléma az volt, hogy ha meg volt adva egy környezeti változó, úgy azt a Backend -> NodeVM -> Python osztálykönyvtár egymásba helyezése során valahol a rendes gépre mentette a ’**/*/tmp’ mappába ’uuid.py’ néven, ahol az ’uuid’ egy egyedi szám. Ezt azonban maga a Python kódot futtató modul nem tudta elérni, így nem volt képes bemeneti paramétert biztosítani a felhasználó kódjának. Erre a megoldás az játék nyelvének a megváltoztatása volt. Az új nyelv a TypeScript lett.

Ezt a nyelvet már tudja futtatni és meg lehet mellé adni környezeti változót, amit el is ér.

3. A Socket.io Backend oldali használatánál előjött egy olyan hiba, aminek következtében bármi WebSocket művelet többször hajtódott végre még úgy is, hogy singleton. Hosszas debugolás után rájöttem, hogy mivel a WebSocketet implementáló osztály használva van egy másik modulban is, így a rossz használata miatt kétszer inicializálódott. A megoldás az volt, a másik service '*.module.ts' fájlban a provide: rész helyett az imports: résznél kellett behúzni a modult.
4. Frontend oldalról volt egy hiba, ahol ha 'useEffect' van használva, azon belül egy 'socket.on()', ha itt kellene egy tömb típusú 'useState'-nek az értékét változtatni, akkor a következő kód helytelen, mert csak a tömb legelső elemét fogja beállítani, amit már a következő megjelenítésnél ki is ürít:

   ```
   1 setChat([...chat, { sender, message }]);
   ```

   Hosszas debuggolás és megoldás keresése után rátaláltam a megoldásra, ami:

   ```
   1 setChat((chat) => [...chat, { sender, message }]);
   ```

5.6 Hosztolás

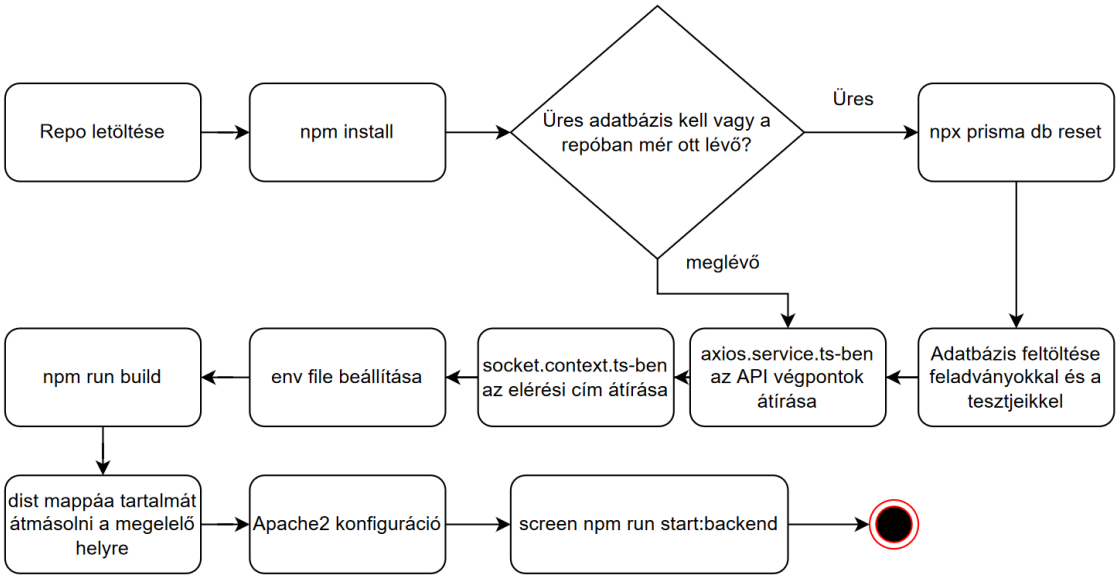
Mivel szeretnék egy olyan oldalt biztosítani a felhasználóknak, ahol bármiféle letöltés és telepítés nélkül tudják használni a szakdolgozat feladatát, ezért az elkészült projektet egy 0/24-ben futó virtuális gépen futtatom. A projekt nevéből kiindulva ezért ez elérhető egy publikus weboldalon[16]. Pár fontos beállítás, amit át kell írni, az a Backend REST API-nak az elérhetősége, hiszen már nem localhost fog kelleni, hanem egy interneten keresztül elérhető cím. Hasonló átírást igényel a WebSocket szervernek az elérhetősége, ami a jelen esetben megegyezik a REST API elérhetőségével. A Frontend már az 'npm run build' script futtatásának az éles környezetre szánt változata elérhető, míg a Backend csak futtatva van, mintha localban lenne. Mivel a gép, ahol fut, az linux, ezért a a 'screen' parancsszóval bővül ki a futtatás parancsa. Az email küldés működése érdekében a környezeti változóba be kell helyeznünk a SendInBlue által szolgáltatott adatokat. Annak érdekében, hogy ez a Backend elérhető legyen az interneten, egy Apache2 konfigban meg van adva, hogy milyen elérhetőségi címre tegye elérhetővé a 'http://localhost:3333/',

[16] https://codeitout.paptamas.com

virtuális gépen futó alkalmazást. Ezt a következő Apache2 virtuális hoszt beállításával el lehet érni:

```
1 <VirtualHost *:80>
2     RewriteEngine On
3     ProxyRequests Off
4     ProxyPreserveHost On
5     ServerName example.com
6     Header set Access-Control-Allow-Origin '*'
7     ProxyPass "/" "http://localhost:3333/" connectiontimeout=5 timeout=30
8 keepalive=on
9     ProxyPassReverse "/" "http://localhost:3333/"
10    ErrorLog ${APACHE_LOG_DIR}/netdata-error.log
11   CustomLog ${APACHE_LOG_DIR}/netdata-access.log combined
12</VirtualHost>
```

Itt fontos, hogy az 5. sorban lévő ’example.com’-ot át kell írni arra a domainre, ahova szántuk a REST API futtatását. Az egész hosztolási folyamat egy ábrán:



Ábra 5.6 Hosztolás folyamata

6 Tesztelés

Egy projektnek fontos része az, hogy a kód részek az elvárt eredményeket adják-e. Amennyiben megfelelően van letesztelve az alkalmazás, úgy a jövőbeni továbbfejlesztésnél elkerülhető az, hogy olyan új kód került a projektbe, ami miatt a már létező kódrész nem az elvárt eredményt adja. Teszteléshez sokféle eszköz és metodológia használható, a szakdolgozat feladatában főleg unit tesztek készültek Backend oldalról és manuális tesztek Frontend oldalról. Ebben a fejezetben bemutatom, hogy az egyes projektrészeknél milyen tesztek készültek, ezeknek mi volt az elvárt eredménye és hogy sikeres volt-e a teszt.

6.1 Backend

Backend oldalon, mint ahogy a 6-os bevezetésben is olvasható volt, unit tesztek készültek. Ezeknek a teszteknek az elkészültéhez egy nagyon népszerű könyvtárosztályt használtam, a 'jest'-et[17]. Miután fel lett telepítve, a még saját service létrehozásakor létrejött egy fájl, aminek a neve '**.spec.ts'. Ebbe lehet a teszteket írni. A könyvtár felkonfigurálása[18] után egy unit teszt így néz ki:

```
1 it('should throw an error if user doesnt exist', async () => {
2   mockCtx.user.findUnique.mockResolvedValue(undefined);
3   await expect(
4     AuthNService.login({ email: 'test@example.com', password: 'test' })
5   ).rejects.toEqual(
6     new HttpException(
7       'User doesnt exist or not activated',
8       HttpStatus.BAD_REQUEST
9     )
10  )
```

A második sorban meg van adva, hogy a 'mockCtx', azaz a mockolt adatbázisnak a '.findUnique()' függvénye milyen értékkel térjen vissza. Mivel ez egy olyan teszt, ahol azt vizsgáljuk, hogy dob-e hibát a bejelentkezés során nem létező emailcímre, ezért egy üres értéket ad vissza. A negyedik sorban meghívjuk az osztálynak azt a metódusát, ami a bejelentkezést kezeli, majd egy feltétel állítunk be az ötödik sortól kezdődően a kimenetelre. A következő táblázat ezeket a unit teszteket foglalja összes olyan formában,

[17] https://www.npmjs.com/package/jest
[18] https://jestjs.io/docs/getting-started

ahol meg van adva a teszt sorszáma, mi a követelménye, a tesztnek egy leírása, mi az elvárt célja a tesztnek, illetve hogy sikeresen lefutott-e.

Sorszám	*Követelmény*	*Tesztleírás*	*Teszt célja*	*Eredmény*
1	Bejelentkezés meghívásakor egy nem létező felhasználónak hibát kell dobni.	A bejelentkezéshez használt interfésszel meghívni a metódust	Egy 400-as hibakódú 'HttpException'-t eldobni	✓
	Bejelentkezés meghívásakor egy még nem aktivált játékosnak hibát kell dobni.	A bejelentkezéshez használt interfésszel meghívni a metódust	Egy 400-as hibakódú 'HttpException'-t eldobni	✓
3	Bejelentkezés meghívásakor egy aktivált felhasználónak hibát kell dobni, ha rossz jelszót adott meg.	A bejelentkezéshez használt interfésszel meghívni a metódust	Egy 400-as hibakódú 'HttpException'-t eldobni	✓
4	Bejelentkezés sikeres meghívásakor egy tokent és a felhasználónevet kell visszaadni.	A bejelentkezéshez használt interfésszel meghívni a metódust	Egy 'UserLoginResult' típusú objektum	✓
5	Regisztráláskor, ha már van használva az email vagy felhasználónév akkor ne dobjon hibát.	A regisztráláskor használt interfésszel meghívni a metódust	Egy Promise<void> visszaadása	✓
6	Regisztrálás sikeres meghívásakor egy üres válasz visszaadása.	A regisztráláskor használt interfésszel meghívni a metódust	Egy Promise<void> visszaadása	✓
7	Felhasználó aktiválásakor hibát dobni, ha nem létezik a felhasználó vagy aktivált.	Az aktiváláshoz használt kóddal meghívni a metódust	Egy 400-as hibakódú 'HttpException'-t eldobni	✓
8	Felhasználó sikeres aktiválásakor egy tokent és a felhasználónevet kell visszaadni.	Az aktiváláshoz használt kóddal meghívni a metódust.	Egy 'UserLoginResult' típusú objektum visszaadása	✓

9	Gyakorló játék indításakor hibát dobni, amennyiben a felhasználó nem létezik.	A felhasználó azonosítójával meghívni a metódust.	Egy 400-as hibakódú 'HttpException'-t eldobni	✓
10	Felhasználó által beküldött kód futtatásakor hibát dobni, ha a játék nem létezik	A metódus meghívása a kód futtatásához szükséges objektummal	Egy 400-as hibakódú 'HttpException'-t eldobni	✓
11	Felhasználó által beküldött kód futtatásakor visszatérni egyből, ha a beküldött kód hibát dob.	A metódus meghívása a kód futtatásához szükséges objektummal	Az addig lefutott teszt eredményekkel egy 'CodeBuildResult' tömbben visszatérése	✓
12	Felhasználó által beküldött kód futtatásakor hibát dobni, ha az egy biztonsági ellenőrzésen nem megy át.	A metódus meghívása a kód futtatásához szükséges objektummal	Egy 400-as hibakódú 'HttpException'-t eldobni	✓
13	Felhasználó által beküldött kód futtatásakor visszatérni az eredményekkel, ha volt legalább 1 nem egyező játékteszt eredmény.	A metódus meghívása a kód futtatásához szükséges objektummal	Egy 'CodeBuildResult' tömbben eltárolt játékteszt eredmények visszaadása, ahol van legalább 1 { correct: false, … };	✓
14	Felhasználó által beküldött kód futtatásakor visszatérni az eredményekkel, ha mind helyes volt.	A metódus meghívása a kód futtatásához szükséges objektummal	Egy 'CodeBuildResult' tömbben eltárolt játékteszt eredmények visszaadása, ahol mindegyik elem {corrent: true, …}	✓

Táblajegyzék 6.1 – Backenden lévő tesztesetek bemutatása

6.2 Frontend

A szakdolgozat 5.6-os befejezésében említett hosztolás után elérhető az interneten a projekt. Mivel erről az oldalról szeretném ugyanazt kapni, mint egy sima felhasználó, ezért manuális tesztelést fogok végrehajtani, mivel ezzel egyben tudom tesztelni a Frontend funkcióinak működését és bármi egyéb vizuális hibát. Ezeknek a teszteknek egy része látható a következő táblázatban.

Sorszám	*Követelmény*	*Tesztleírás*	*Teszt célja*	*Eredmény*
1	A bejelentkezési oldalon a kis betűben megjelenített aloldalak között lehet oda és vissza lépkedni.	Bármelyik kékkel jelölt link elvisz a hozzátartozó másik oldalra.	Ne dobjon hibát, amennyiben a gombok vannak használva.	✓
2	Nem létező felhasználói adatok megadása el elküldése után az eredmény megjöveteléig a gombnak ki kell kapcsolódnia.	A mezőket kitöltése, majd az elküldés gomb megnyomása.	Egyszerre csak 1 bejelentkezési kérést lehessen küldeni.	✓
3	Nem létező felhasználói adatok megadása el elküldése után az eredmény után egy hibaüzenetet kell megjelenítenie.	A mezőket kitöltése, majd az elküldés gomb megnyomása, a válasz megvárása.	Kapjon visszajelzést a felhasználó, hogy sikertelen volt a bejelentkezés.	✓
4	Sikeres bejelentkezés után a főoldalra kell irányítani a felhasználót.	A mezőket kitöltése, majd az elküldés gomb megnyomása, a válasz megvárása	Ne kelljen a felhasználónak manuálisan a főoldalra navigálnia.	✓
4	Főoldalon a gyakorlás menüből a játék kezdés gomb megnyomására szinte azonnal a játék oldalára irányít.	A gomb megnyomására az oldal elirányítóik.	A játék indulását automatikusan kapja meg a felhasználó.	✓

5	Játék oldalon legalább 1 rossz, de legalább 1 jó teszt futtatása után ki kell zöldülnie a teszteset dobozának, amelyik a jóhoz tartozik.	A ’Build’ gomb megnyomására, majd a válasz megérkezésének megvárása.	A felhasználónak látnia kell mennyi teszteseten ment át az általa megadott kód.	✓
6	Játék oldalon amennyiben minden teszten átment a kód, egy felugró ablaknak kell megjelennie.	A ’Build’ gomb megnyomására, majd a válasz megérkezésének megvárása, amire egy felugró ablak jelenik meg.	A felhasználónak látnia kell biztosra, hogy az általa írt kód megfelelő.	✓
7	A játék oldalon a játék végét jelző felugró ablak gombjának vissza kell vinnie a főoldalra.	A felugró ablakban meg kell nyomin a ’Home’ gombot.	Meg kell könnyíteni az oldalak közötti navigációt.	✓
8	Egy többjátékos oldalon az enter lenyomására a chat ablakba beírt üzenet elküldése kerül és megjelenik a csevegés listában.	A chat ablak input mezőjébe egy szöveget kell írni, majd egy enter nyomni.	Egyszerűen lehessen a csevegés ablakba írni, ne kelljen az elküldés gombot megnyomni.	✗
9	Egy többjátékos oldalon az az elküldés gomb lenyomására a chat ablakba beírt üzenet meg kell jeleníteni.	A chat ablak input mezőjébe egy szöveget kell írni, majd a ’Send’ gombra nyomni.	Lehessen a csevegés ablakba üzenetet küldeni gombnyomásra.	✓

Táblajegyzék 6.2 – Frontenden lévő tesztesetek ismertetése

7 Továbbfejlesztési lehetőségek

A szakdolgozat feladatát több irányba is tovább lehet fejleszteni, jobbá tenni. Ezek közül ilyen lehet:

- A játék oldalon elérhetővé tenni több másik programozói nyelvet, amit a játékosok használhatnak. Egy új nyelv - ami nem jár új osztálykönyvtár telepítésével - lehet a JavaScript.
- Annak érdekében, hogy biztonságosabb legyen a hitelesítés és engedélyezés:
    - a JWT tokent titkosító kulcs áthelyezése egy környezeti változóba.
    - Lekezelni, hogy a felhasználó csak a saját játékára tudjon megoldást adni.
- A jelenlegi Frontend oldal kinézete mellé egy újat létrehozni, ami a látássérült játékosoknak megkönnyítheti a weboldal használatát.
- A felhasználók moderálása érdekében lehetőséget biztosítani arra, hogy üzeneteket lehessen törölni, felhasználókat lehessen kitiltani.
- Amennyiben a játékosok elhagynák a játék oldalát és az még nem ért véget, úgy a visszacsatlakozást követően az utolsó beküldött kódot töltse vissza.
- Amennyiben egy játékos beküldött egy helyes megoldást, lehetőséget biztosítani egy olyan opcionális gombra, aminek megnyomásával megmutathatja az ellenfelének, hogy ő hogyan oldotta meg azt a bizonyos feladatot.
- Dicsőségtáblát készíteni a feladatokhoz, ahol meg lehet nézni, hogy mely felhasználók oldották meg leggyorsabban.
- Új játékmódok biztosítása, például: csapatjátékok, több feladatból álló versenyek.
- Az éles rendszer futtatásának biztonságosabbá tétele, ne egyből a fő rendszeren fusson, hanem egy dockerizált környezetben.
- A nagyobb forgalom kezelése érdekében ezt a dockerizált változatot egy kubernetes környezetben futtatni, ami automatikusan tudna új Backend és Frontendet létrehozni és törölni.

8 Összegzés

A szakdolgozat feladata annak érdekében készült el, hogy a 2.1-es hasonló rendszerek bekezdésben bemutatott weboldalaknak legyen egy olyan alternatívája, ami nem a végtelenségig van bonyolítva, hanem csak a lényegre fókuszál, ami nem más, mint hogy játékosan tanuljanak meg az emberek programozni. Ezt szerintem sikerült elérnem, mivel sikeresen készítettem egy weboldalt, ami megállja a helyét a hasonló alkalmazásokkal szemben. Ennek érdekében kellett választanom olyan keretrendszereket mind Backend, mind Frontendhez, ami hozza a mai elvárt funkcionalitásokat. Ehhez került kiválasztásra a NestJs és a React, ami véleményem szerint a fejlesztés során tökéletes választásnak minősült. A weboldal képes egy játékosnak a kódját fogadni, azt el tudja küldeni a Backendnek egy gomb megnyomására, ott pedig egy jól kiválasztott függvénykönyvtárnak köszönhetően biztonságos környezetben le is lehet futtatni. Azonban a kód futtatása nem elég, meg kell bizonyosodni arról, hogy a játék feladatára helyes megoldás, ezért egy jól létrehozott logika segítségével ezt egy előre meghatározott tesztesetek ellen lehet futtatni. A játék oldal kód szövegdobozához olyan osztálykönyvtár került, ami könnyedén tudja támogatni a másik nyelvek szintaktikáját egy esetleges továbbfejlesztés esetén. A szövegdobozban van olyan alap funkció, ami mutatja a lehetséges kód kiegészítését. Ez szerintem egy ilyen rendszerben elvárás is lenne, ezért ezen a téren előrébb van pár hasonló rendszerénél a projekt feladata. Ez az új játékosoknak hatalmas segítség, hiszen ők még nem biztos, hogy tudják a nyelv specifikus metódusokat egy objektumhoz például. A játékosoknak nem kell azon aggódniuk, hogy nem az ő szintjükhöz igazodó feladatot kapnak, hiszen a szakdolgozat feladata során sikerült egy olyan logika kialakítása, ami ezt biztosítja. Ezzel párhuzamban van az a logika, ami nem privát többjátékos mód esetén a játékos tudásához legközelebbi felhasználót sorsolja be, ezzel egy igazságos játékélményre van lehetőség a felhasználók felé. A játékosok közötti kommunikációt sikerült letisztultan és egyszerűen megoldani. A feladat során több nehézség és akadály jött szembe, ezekre valamilyen formában sikerült megoldást találni. Fontos volt nekem, hogy a feladat legyen elérhető bárki számára egy publikus weboldalon már futtatva az átlagembernek is, pont, akiknek ez a játék szól.

9 Absztrakt

A szakdolgozat elején be lett mutatva egy probléma, azaz, hogy nincs egy olyan egyszerű és lényegre törő weboldal, ami nem fókuszál másra, mint hogy a felhasználóit tanítsa meg programozni játékos módon. Pár már létező rendszert bemutattam, ami a leginkább hasonlít az én általam elképzelt weboldalhoz. Ezeknek az internetes oldalaknak a főbb funkcióikon végig mentem, ahol bemutattam, hogy ott mit csináltak meg kiválóan és min lehetne még fejleszteni. Hogy a saját weboldal meg is valósuljon, szükséges keretrendszereket választani, amiknek a bemutatása után ki is választottam a Backendhez a NestJs-t, illetve a React-ot a Frontendhez. Itt azt is eldöntöttem magyarázattal, hogy a játékosok miért TypeScript programnyelvben fognak játszani. Hogy ne a fejlesztés közben kelljen kitalálni, hogy pontosan milyen funkciók kellenek, ezért külön fejezetet szenteltem annak, hogy egy pontos elvárásokkal teli listát fogalmazzak meg. Ezek után különböző ábrákkal szemléltettem a szakdolgozatban azt, hogy a funkciók milyen kisebb lépésekből állnak, mi mit követ, hogyan kapcsolódik a Backend és a Frontend, melyik rétegben kapcsolódik a Backend az adatbázishoz stb. Ezt követték a szakdolgozatnak a gyakorlatiasabb részei, a fejlesztés és tesztelés fejezetek. A fejlesztés bevezetése után két fő egységre bontottam a szakaszt. Backend oldalról bemutattam kódrészletekkel szemléltetve a főbb funkcióknak a működését, hogyan lehet a játékosok közötti valós idejű kommunikációt megoldani a Socket.io segítségével, mi módon lehet a játékosokat hitelesíteni és engedélyezni JWT tokenekkel, milyen módon lehet egy más által írt kódot egy biztonságos környezetben lefuttatni a vm2 függénykönyvtárral, milyen funkcionalitás készült arra, hogy a lefuttatott kódot miként lehet letesztelni, hogy valóban jó megoldások-e stb. Frontend oldalról bemutatásra került az, hogy ezeket a funkciókat hogyan implementáltam, hogyan értem el azt, hogy csak a bejelentkezett játékosok tudják használni a rendszert, hogyan néznek ki az aloldalak és ott pontosan miket lehet csinálni. Ezeket követte a tesztelés fejezet, ahol a megírt funkciókat teszteltem unit és manuális teszteléssel, hogy valóban azt csinálják-e, amit kell. A szakdolgozat végét egy összegzés zárta, ahol saját vélemény alapján elmondtam, hogyan sikerült a feladat teljesítése, elérte-e azt a weboldal, aminek a célja lett volna.

10 Abstract

At the beginning of the thesis, a problem was presented, namely that there is no simple and meaningful website that does not focus on anything other than teaching its users to program in a playful way. I presented a few existing systems that most closely resemble the website I envisioned. I have gone through the main features of these websites, showing what they have done excellently and what could be improved. In order to make my own website, it is necessary to choose frameworks, which after presenting them, I chose NestJs for the Backend and React for the Frontend. Here I also decided to explain why the players will play in TypeScript programming language. In order to avoid having to figure out exactly what features I need during development, I dedicated a separate chapter to formulating a list of precise requirements. After that, I used different diagrams in the thesis to illustrate how the functions consist of smaller steps, what follows what, how the Backend and the Frontend are connected, in which layer the Backend is connected to the database, etc. This was followed by the more practical parts of the thesis, the development and testing chapters. After the introduction of development, I divided the section into two main units. On the Backend side, I demonstrated with code snippets how the main features work, how real-time communication between players can be solved using Socket.io, how players can be authenticated and authorized with JWT tokens, how code written by someone else can be run in a secure environment using the vm2 package, what functionality was created to test how the code run is actually good solutions, etc. From the Frontend side, how I implemented these functionalities, how I achieved that only logged in players can use the system, how the subpages look like and what exactly can be done there. These were followed by a testing chapter where I tested the features I had written by unit and manual testing to see if they actually did what they were supposed to do. The thesis ended with a summary, where I gave my own opinion on how the task was completed and whether the website achieved what it was intended to achieve.

11 Irodalomjegyzék

[1] Banks, A., Porcello, E.: Learning React. O'Reilly Media, Inc,. 2017

[2] Gackenheimer, C.: What Is React? Apress, Berkeley, CA, 2015

[3] Wohlgethan, E.: Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue.js. Hochschule für angewandte Wissenschaften Hamburg, 2018

[5] Tilkov, S., Vinoski, S.: Node.js: Using JavaScript to Build High-Performance Network Programs, IEEE Internet Computing, vol. 14, no. 6, pp. 80-83, Nov.-Dec. 2010

[6] Subecz, Z.: WEB-DEVELOPMENT WITH LARAVEL FRAMEWORK, Gradus Vol 8, No 1 (2021) 211-218, 2021

[7] Fette, I., Melnikov, A.: https://datatracker.ietf.org/doc/html/rfc6455, utoljára megtekintve: 2021. 12.05

[8] Mysliwiec, K.: https://docs.nestjs.com/, utoljára megtekintve: 2021. 12. 08

12 Ábrajegyzék