



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

Informatikai Kar

Programozáselmélet és Szoftvertechnológia Tanszék

Function Wars

Várkony Teréz Anna

Egyetemi adjunktus

Torma Levente Péter

Z4TNSZ

Program Tervező

Informatikus

Budapest, 2023

1 Tartalomjegyzék

1	Tartalomjegyzék	1
2	Rövidítések	3
3	Bevezetés	4
4	Felhasználói dokumentáció	6
4.1	Regisztráció/Belépés	6
4.2	Regisztrált felhasználók számára	8
	Főoldal	8
	Barátokhoz kapcsolódó menü pontok	10
	Csoportos chat/chat szoba	12
	Játék létrehozás	13
4.3	Vendégeknek	15
	Csoportos chat	16
4.4	Játék (vendég- és regisztrált felhasználóknak)	17
4.5	Pálya lista, pálya szerkesztés (adminoknak és felhasználóknak)	21
	Pályalista	21
	Pályaszerkesztő	22
4.6	Adminoknak és a szuper adminnak	24
	Felhasználók listában	25
	Jelentések listája	25
	Adminok listája	26
4.7	Szuper adminoknak	27
5	Fejlesztői dokumentáció	35
5.1	Backend	35
	Adatbázis	35
	API	42
	Websocketek	60
	Segéd osztályok (utils)	63
	Használt külső csomagok	69
5.2	Frontend	70
	Fontosabb komponensek	72
5.3	Tesztelés	73
6	Összefoglalás	74
7	Irodalomjegyzék	75

2 Rövidítések

ORM – Object Relational Mapping

API – Application Programming Interface

JWT – JSON Web Token

HTTP – HyperText Transfer Protocol

JS – JavaScript

TS – TypeScript

UUID – Universally Unique Identifier

JSON – JavaScript Object Notation

SQL – Structured Query Language

TLS – Transport Layer Security

CLI – Command Line Interface

DOM – Document Object Model

SMTP – Simple Mail Transfer Protocol

3 Bevezetés

Egy többjátékos játékot valósítottam meg, melynek célja, hogy egy játékos eltaláljon valakit egy matematikai függvénnyel. A játékot 2-4 játékos játszhatja és a játék akkor ér véget, ha egy függvény képe eltalált egy játékost. Minden játékos csak olyan függvényeket adhat meg, amelyeknek nincs szakadás pontja, tehát pl. $\frac{a}{x}$ alapú függvény nem adható meg. A függvényeket lehet kombinálni pl. $\sin x + \frac{2x}{3}$. A függvényeket a felhasználók a használható gombokon láthatják. A függvények ellenőrzése kliens- és szerveroldalon is megtörténik, alapvetően reguláris kifejezések segítségével vannak ellenőrizve, de ellenőrizve van az is, hogy van-e olyan pont, ahol nem értelmezett a függvény, amihez 0-tól elkezd ki számolni a függvény értékeit, pozitív és negatív irányba is. Minden játékos esetén a (0;0) koordináta a saját „bázisuk” középpontja.

Grafikus szerkesztővel többféle pályát is létre lehet hozni. Lehetőség van választani 2 féle akadály közül (ellipszis és téglalap). Ezek magassága és szélessége állítható, „drag-and-drop”-val pedig mozgatható a pályán. Ezek az akadályok blokkolják a függvény képét és amennyiben ez bekövetkezett, akkor az akadály sérül – a függvény által megtett út hosszától függően – ezzel szépen lassan el lehet őket pusztítani. A játékosok számát is meg lehet adni, ez min. 2 és max. 4 lehet, a játékosok bázisát az akadályokhoz hasonlóan „drag-and-drop”-val lehet mozgatni, de a méretük rögzített.

A játékosoknak van lehetősége egymással beszélgetni, mind privát üzenetváltásban, mind a játékban lévő játékosokkal. Ahhoz, hogy ez a folyamatos kommunikáció minden játékos között létrejöjjön socket-eket használok, amikről a Telekommunikációs hálózatok tárgyon tanultunk. Socket-ek kezeléséhez SocketIO-t használok, melyet Szerveroldali webprogramozáson tanultam.

A felhasználók 4 féle jogosultsági kör egyikébe tartozhatnak: szuper admin, admin, regisztrált felhasználó (továbbiakban csak felhasználó) és vendég. A felhasználók tudnak játékhoz csatlakozni, saját játékot és pályát létrehozni, más játékosokkal barátságban lenni, és velük privát üzeneteket váltani. A vendég felhasználók csak játékhoz tudnak csatlakozni, és a csoportos chat funkciót használni. A szuper admin felhasználó automatikusan jön létre, ezzel a felhasználóval lehet sima admin felhasználókat kinevezni. Az adminok a játék funkcióhoz nem férnek hozzá, csak karbantartó funkciókhoz, pálya létrehozás és törlése, jelentések és felhasználók kezelése funkciókhoz. A felhasználókat ki tudják tiltani, chat üzeneteiket blokkolni (privát üzenetek kivételével), illetve minden admin tud adni és elvenni admin jogot.

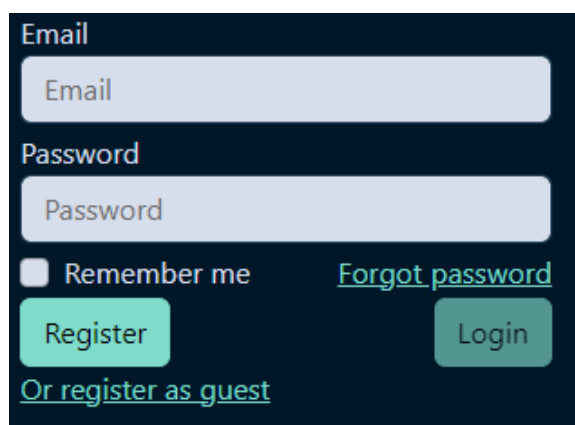
A szerver- és a kliensoldal teljes mértékben különálló. A szerver API-ként működik és a kliens HTTP kéréseket küld a szervernek. A szerver Type- és JavaScript-ben íródott NodeJS-t használva, a TypeScript kódok (JavaScript kódokkal együtt) egy külön mappába fordulnak le futtatás előtt. A kéréseket a kliens felől express szerver segítségével kezelem. MySQL adatbázist futtatok ennek kezelésére Sequelize ORM keretrendszerrel használom. Egy ORM keretrendszer segít az adatbázist objektum elvűen feldolgozni, ez annyit jelent, hogy minden táblához van egy modell, ami a programon belül reprezentálja a táblát, és tartalmazza a tábla kezeléséhez tartozó metódusokat. Továbbá egy migráció, melynek feladata a táblák létrehozása, ha még nem lettek létrehozva.

4 Felhasználói dokumentáció

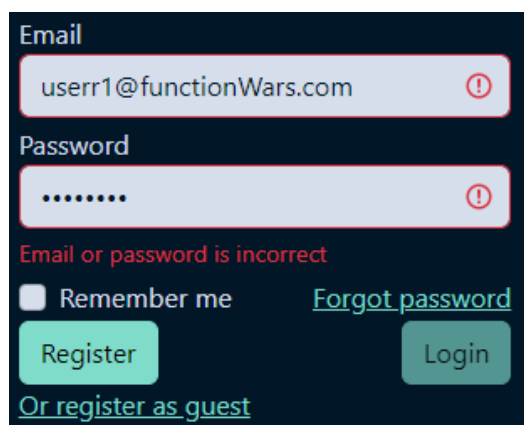
Az oldalt érdemes teljes képernyőben megnyitva látogatni, ugyanis bizonyos elemeket nem lehet minden ablak méretben megjeleníteni. Kisebb méretek esetén ilyenkor görgetni kell – jobbra, balra és/vagy le, föl –.

4.1 Regisztráció/Belépés

Az oldalra lépve, amennyiben a bejelentkezés még nem történt meg megjelenik egy belépés felület (ld. ábra 4.1:1). Ha egy felhasználó már beregisztált, akkor az e-mail címével és jelszával tud lépni, illetve a felhasználó kérheti, hogy belépve szeretne maradni („Remember me” ld. ábra 4.1:1). Előfordulhat, hogy valaki rosszul írja be az adatait, ilyenkor az oldal hibaüzenetet ír ki, hogy valamilyen adat hibás („E-mail or password is incorrect” ld. ábra 4.1:2).



ábra 4.1:1



ábra 4.1:2

Abban az esetben pedig, ha valaki elfelejtette a jelszavát, lehetősége van újat megadni, ehhez a „Forgot password” (ld. ábra 4.1:3) opciót kell választani. Majd meg kell adni az e-mail címet (ld. ábra 4.1:4), itt is szól az oldal, ha az e-mail nem volt megfelelő (ld. ábra 4.1:5), illetve jelez abban az esetben is, ha sikeres volt (ld. ábra 4.1:6). Sikeres kitöltés esetén egy e-mailt küld a szerver a felhasználó e-mail címére (ld. ábra 4.1:7). Az e-mailben található gombra kattintva lehet új jelszót beállítani – gomb alatt látható a link, amire irányít a gomb – (ld. ábra 4.1:8). A jelszónak legalább 8 karakter hosszúnak kell lennie, és meg kell erősíteni (Hibaüzeneteket ld. ábra 4.1:9). Miután sikeres volt az új jelszó kérés a felhasználó beléphet az új jelszával. Fontos továbbá, hogy ezekre a jelszó helyreállító e-mailekre idő korlát van, és 1 óra után már érvénytelenekké válnak.

Email

Password

☐ Remember me [Forgot password](#)

Register Login

[Or register as guest](#)

ábra 4.1:3

Email

Send email

ábra 4.1:4

Email

userr1@functionWars.com

Email not found

Send email

ábra 4.1:5

Email sent

Ok

ábra 4.1:6

Dear Dessie36

As you requested your reset password link has been created. Please click on the button below (or directly to the link)

Reset password

<http://localhost:4200/reset-password/078a3544-b475-4a3d-a952-dba0d7ea2022>

ábra 4.1:7

Password

Confirm password

Reset my password

ábra 4.1:8

Password

.....

Password must be at least 8 characters

Confirm password

.....

Passwords do not match

ábra 4.1:9

Regisztráció esetében meg kell adni egy nevet, e-mail címet és jelszót (ld. ábra 4.1:10). A névnek és az e-mail címnek egyedinek kell lennie, tehát amennyiben más felhasználó már használja valamelyiket akkor mást kell megadni. Név továbbá 3 és 20 karakter hosszúság között lehet. Jelszót minden esetben meg kell erősíteni, és minimum

8 karakter hosszúnak kell lennie (néhány minta hibáért ld. ábra 4.1:11). Regisztráció közben lehet kérni, hogy az oldal megjegyezze az adatokat, amik ahhoz szükségesek, hogy belépve maradjon a felhasználó. Ehhez a „Remember me” opciót kell kipipálni (ld. ábra 4.1:10).

Name

Email

Password

Password confirmation

☐ Remember me

Login Register

ábra 4.1:10

Name

1234567891234567891234

Name must be maximum 20 character

Email

Email

Email is required

Password

.....

Password must be at least 8 character

Password confirmation

.....

Passwords are not same

☐ Remember me

Login Register

ábra 4.1:11

Abban az esetben, ha egy felhasználó nem szeretne regisztrálni, akkor erre is van lehetőség, az „Or register as guest” linkre kattintva egy név megadásával be lehet lépni, erre a névre ugyanazok a követelmények igazak, mint regisztráció esetén.

Name

Name

Login as guest

ábra 4.1:12

Name

2

Name must be at least 3 characters long

Login as guest

ábra 4.1:13

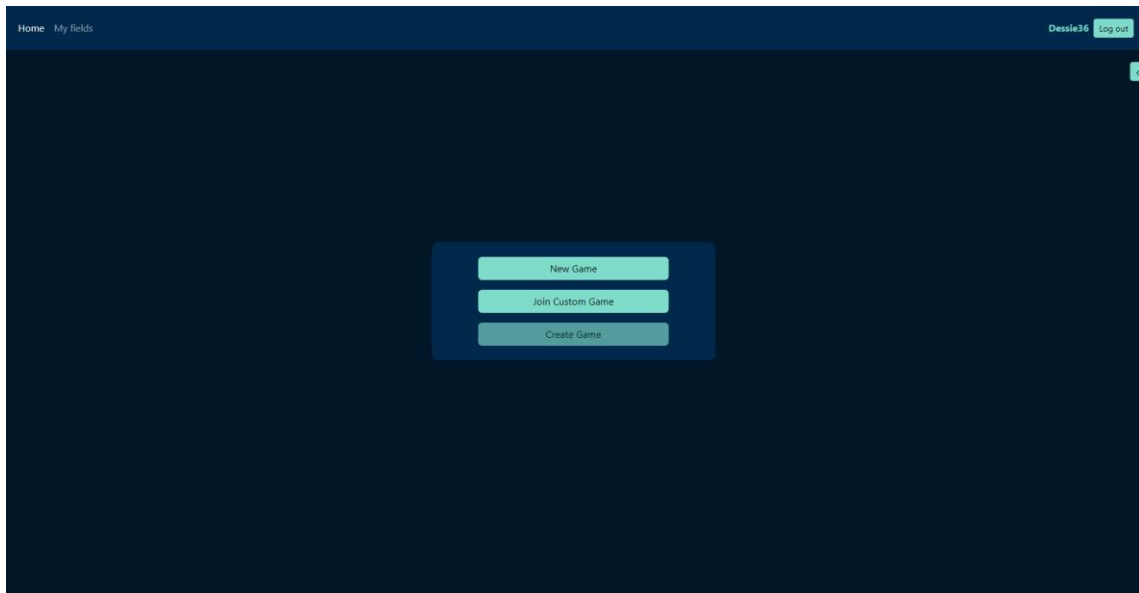
4.2 Regisztrált felhasználók számára

Főoldal

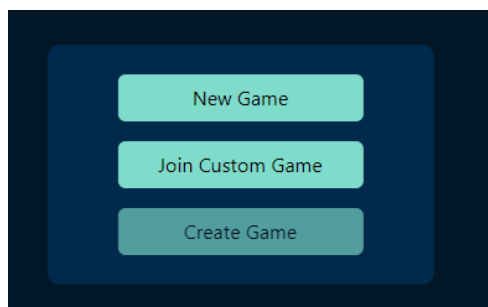
Belépés után a főoldalon (fő oldalt teljesen ld. ábra 4.2:1) található egy mini menü (ld. ábra 4.2:2), ahol a játék megkezdéséhez találhatóak gombok. Továbbá van egy navigációs sáv (ld. ábra 4.2:3), és egy kinyitható fül (ld. ábra 4.2:4 és ábra 4.2:5) – egeret fölé kell vinni, hogy kinyíljon –. A kinyitható füllel megtekinthető a barátok listája és más

barátokhoz kapcsolódó menüpontok is itt találhatóak (ld. Barátokhoz kapcsolódó menüpontok). Amennyiben a felhasználónak van egy új értesítése, akkor egy kis piros pont jelenik meg (ld. ábra 4.2:6).

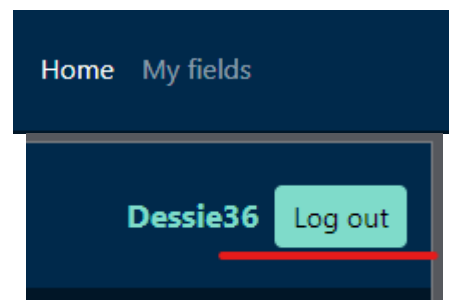
Kijelentkezni a navigációs sávban látható „Log out” gombbal lehet (ld. ábra 4.2:3).



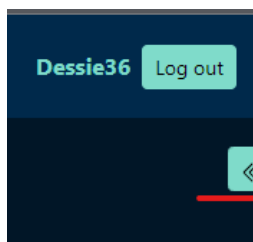
ábra 4.2:1



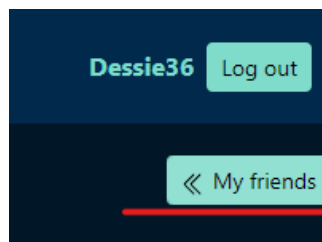
ábra 4.2:2



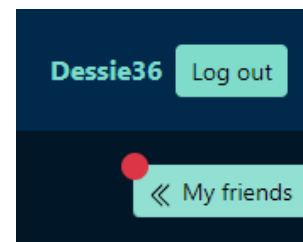
ábra 4.2:3



ábra 4.2:4 (zárt állapot)



ábra 4.2:5 (nyitott állapot)

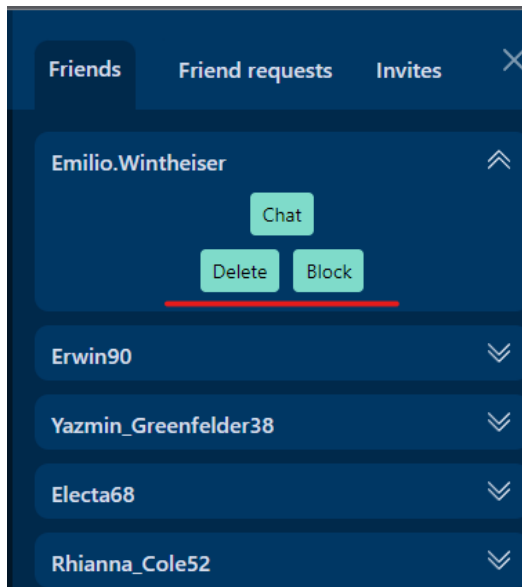


ábra 4.2:6

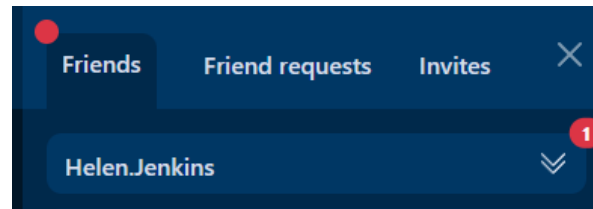
Barátokhoz kapcsolódó menü pontok

A barátok listában („Firends”) lehetőség van a barátoknak üzenetet küldeni („Chat”), törölni valakit („Delete”), illetve blokkolni („Block”), mely egyben törli is a barátot (ld. ábra 4.2:7). Ahhoz, hogy egy baráthoz elérhesse a felhasználó ezeket az opciókat, rá kell kattintani egy barátra. Azonban nem szükséges a kis nyilakra klikkelni, az adott barátnál bárhova lehet annak érdekében, hogy lenyíljon vagy összecsukódjon. Ha egy barátnál olvasatlan üzenet van, akkor egy kis piros pötty jelenik meg az adott barátnál a jobb felső sarokban, egy számmal a belsejében, hogy mennyi olvasatlan üzenet van (ld. ábra 4.2:8).

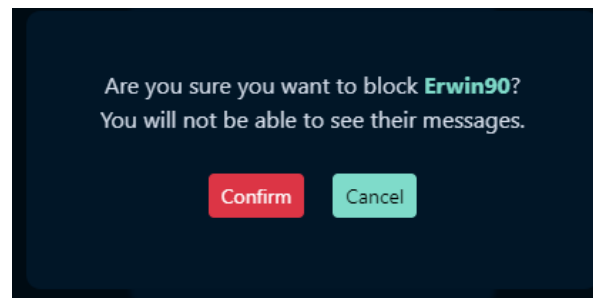
Törlés esetén minden esetben fel ugrik egy megerősítő „pop-up”, a „Confirm” gombra kattintva a barát törlésre kerül, és természetesen a „Cancel” pedig bezárja a „pop-up”-ot (ld. ábra 4.2:9). A blokkolás esetében is egy ugyan ilyen megerősítő „pop-up” jön elő. Blokkoláskor is törlődik a barát, de ez annyiban más, hogy későbbiekben nem fogja látni az üzeneteit a csoportos chatben.



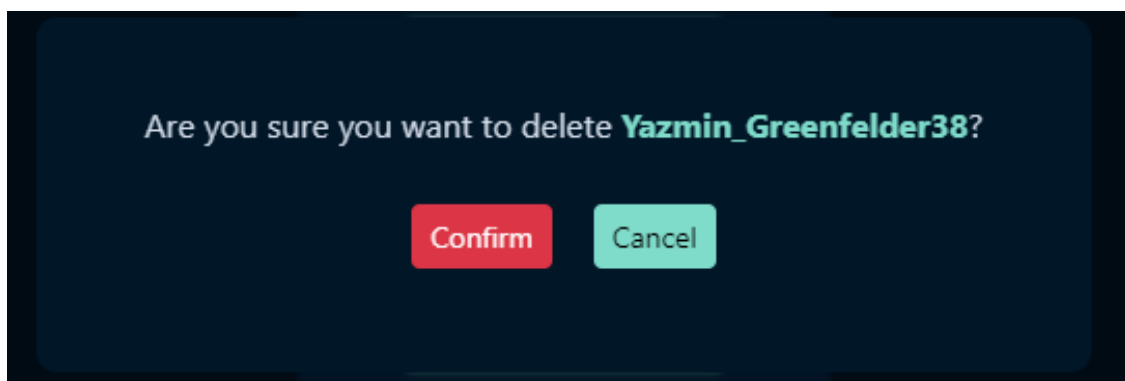
ábra 4.2:7



ábra 4.2:8

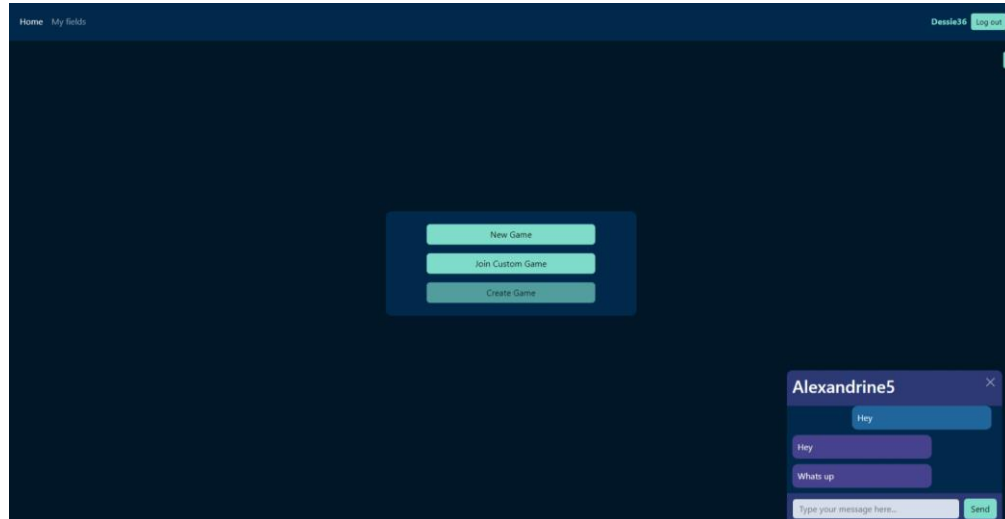


ábra 4.2:9

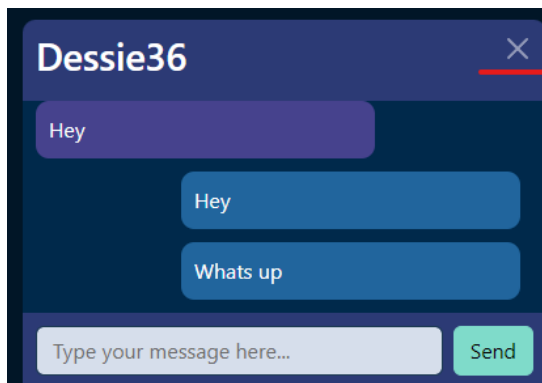


ábra 4.2:10

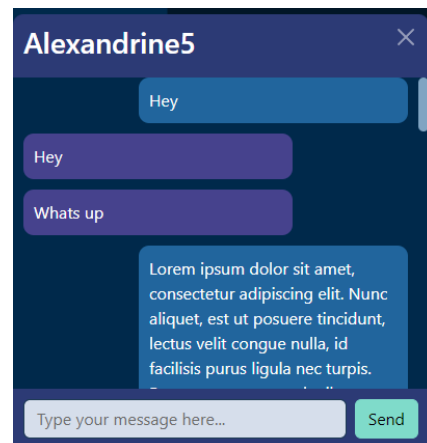
A „Chat” gombra kattintva megjelenik egy chat ablak a jobb alsó sarokban (ld. ábra 4.2:11). A saját üzenetek jobb oldalt találhatók kék háttérrel, és a barát üzenetei pedig lilával (ld. ábra 4.2:12 és ábra 4.2:13). Üzenetet küldeni a „Send” gombra kattintva lehet, vagy az enter megnyomásával. Chat ablakot a jobb felső sarokban lévő „X” ikonra kattintva lehet bezárni (ld. ábra 4.2:12).



ábra 4.2:11

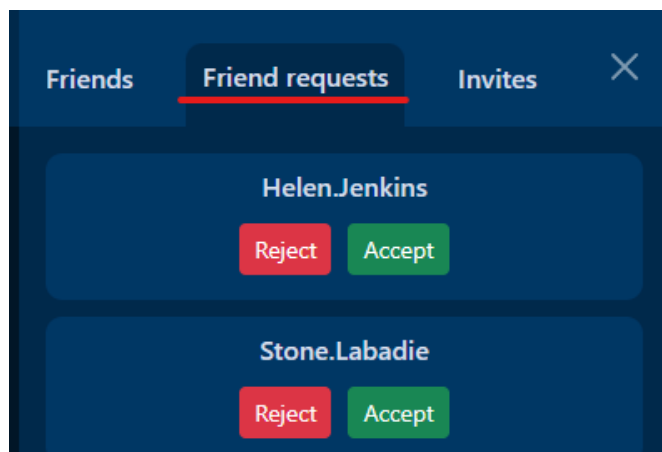


ábra 4.2:12



ábra 4.2:13

A barát kérelmek a „Friend requests” menüpontban találhatók, a „Reject” gombbal el lehet utasítani és az „Accept” gombbal pedig el lehet fogadni. Elfogadás után rögtön látható lesz a barátok listában (ld. ábra 4.2:14).

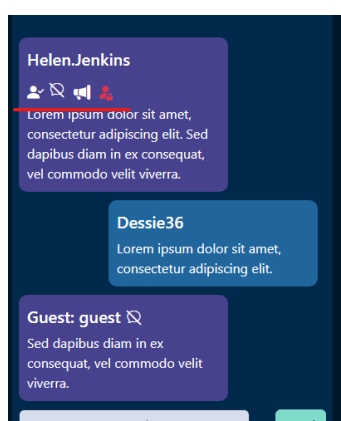


ábra 4.2:14

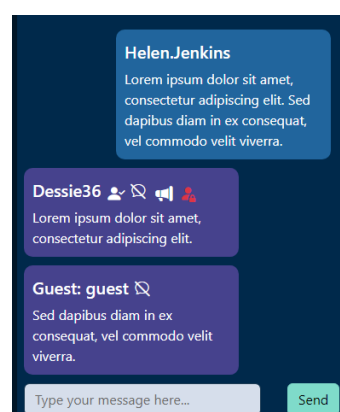
Az „Invites” menüpontban látható a játékba való meghívások. Formailag hasonlóan néz ki, mint a „Friend requests”, de egy másik fejezetben lesz erről szó részletesebben.

Csoportos chat/chat szoba

A csoportos chat ugyan úgy néz ki, mint a privát chat, csak több funkcióval van ellátva. Minden üzenetnél vannak ikonok, amikkel az adott felhasználóval/vendéggel lehet műveleteket végre hajtani (ld. ábra 4.2:15 és ábra 4.2:16). Minden felhasználó hozzáfér minden funkcióhoz a játékban és a várószobában lévő chat-nél is. Ezek a barátoknak adás, némítás, jelentés és blokkolás, mindegyiket egy kis ikon reprezentálja (ld. ábra 4.2:15). Amennyiben egy felhasználónak nem egyértelmű, hogy melyik mit jelent, akkor, ha fölé viszi az egeret, akkor „tooltip”-ként szövegesen is megjelenik, hogy mit csinál (ld. ábra 4.2:17). Minden esetben, miután egy ikonra lett kattintva, felugrik egy kis ablak, mely vissza jelzést ad, hogy sikeres volt a művelet (ld. ábra 4.2:18). A jelentés bizonyos tekintetben kivételt képez, ekkor először egy olyan ablak ugrik fel, ahol meg kell adni a jelentés okát, ez kötelező és csak ezután ugrik fel az ablak, hogy sikeres volt (ld. ábra 4.2:19).

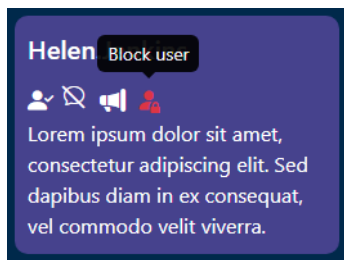


ábra 4.2:15

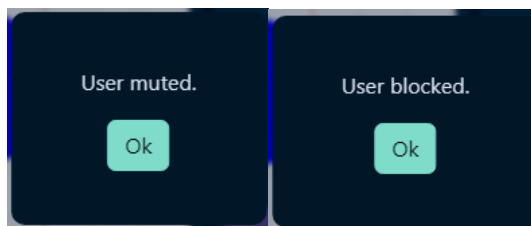


ábra 4.2:16

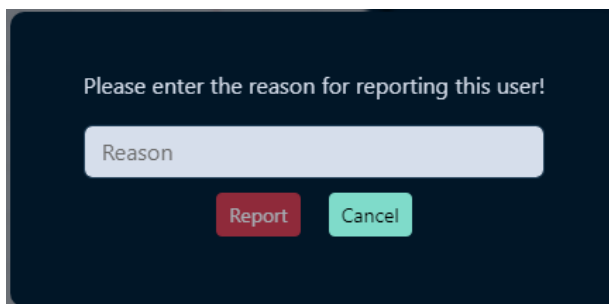
¹ Sógó az adott elemhez, hogy az mit csinál.



ábra 4.2:17



ábra 4.2:18

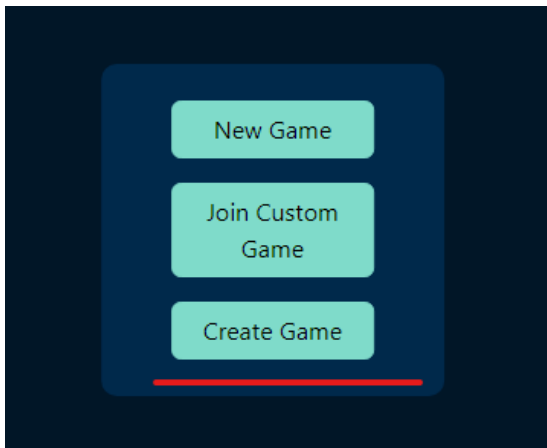


ábra 4.2:19

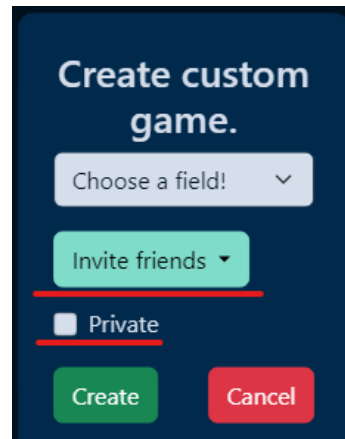
Játék létrehozás

A felhasználók a létre hozott pályáikkal indíthatnak saját játékokat, ehhez a „Create Game” gombra kell kattintani (ld. ábra 4.2:20). Majd fel ugrik egy ablak (ld. ábra 4.2:21), ahol ki kell választani, hogy melyik pályával szeretne játékot indítani. Itt van lehetőség barátokat meghívni az „Invite friends” gombra kattintva (ld. ábra 4.2:22), ahol lehetőség van meghívni az online barátoakt. Továbbá lehetőség van privátra állítani a játékot, tehát csak a meghívott barátok csatlakozhatnak (ld. ábra 4.2:21). A meghívott barátok egy kis piros pöttyöt fognak látni a „My Friends” fülön, és a menüben pedig az „Invites” menüpontnál is megjelenik egy piros pötty (ld. ábra 4.2:23).

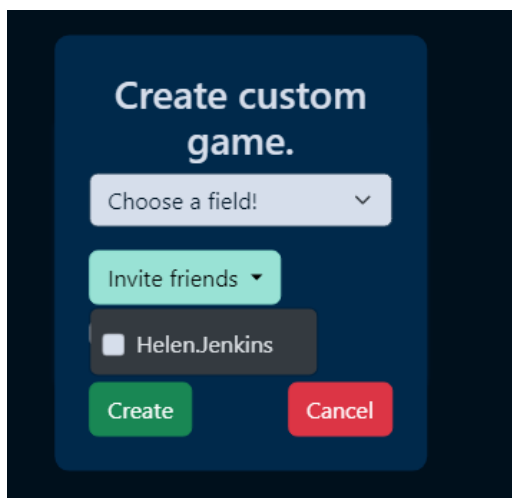
Miután létre lett hozva a játék, egy várószobába kerül a tulajdonos (ld. ábra 4.2:24). A „Cancel custom game” gomb törli a játékot és abban az esetben, ha már csatlakozott több játékos is, akkor megjelenik azoknak a játékosoknak egy üzenet (ld. ábra 4.2:25). A várószobában lévő játékosok száma is látható, illetve, ha megtelt a várószoba, akkor a „Start Game” gombbal lehet elindítani a játékot (ld. ábra 4.2:24). Ha egy játékos csatlakozott a várószobához, akkor a tulaj a csoportos chat felett látni fogja, és ki tudja őket rakni a várószobából (ld. ábra 4.2:26).



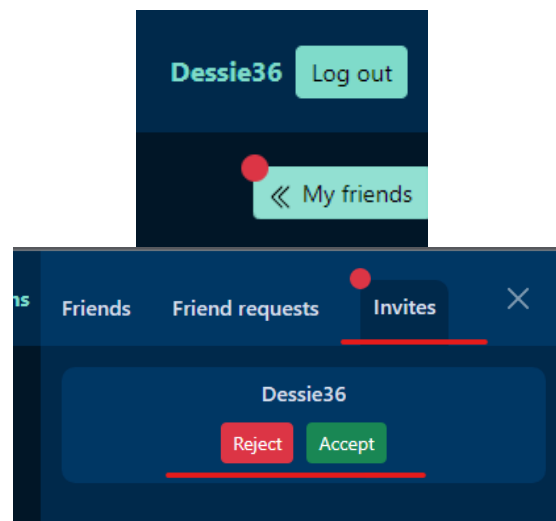
ábra 4.2:20



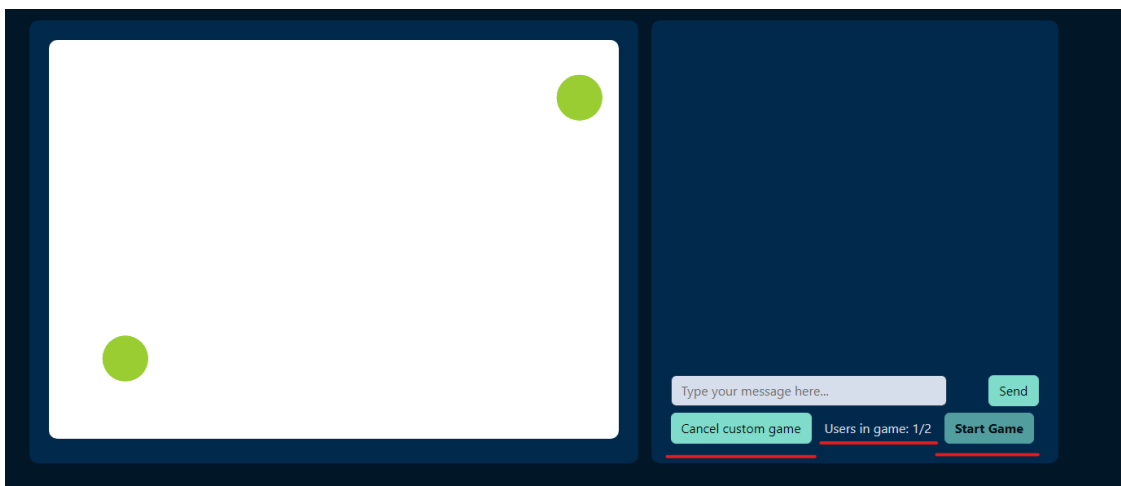
ábra 4.2:21



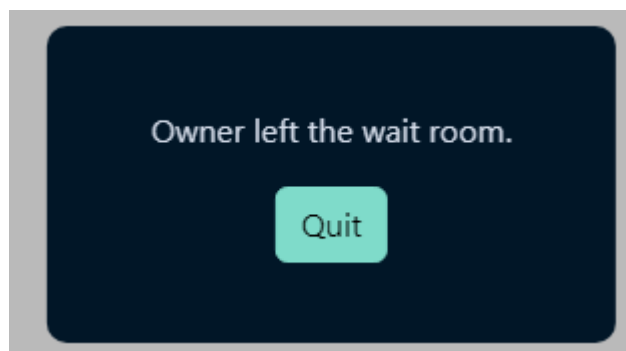
ábra 4.2:22



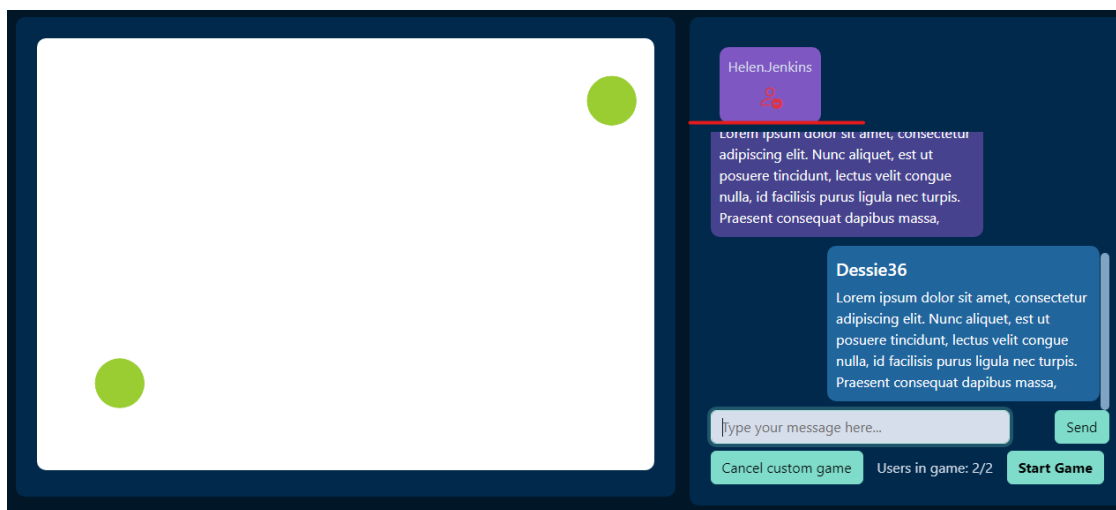
ábra 4.2:23



ábra 4.2:24



ábra 4.2:25



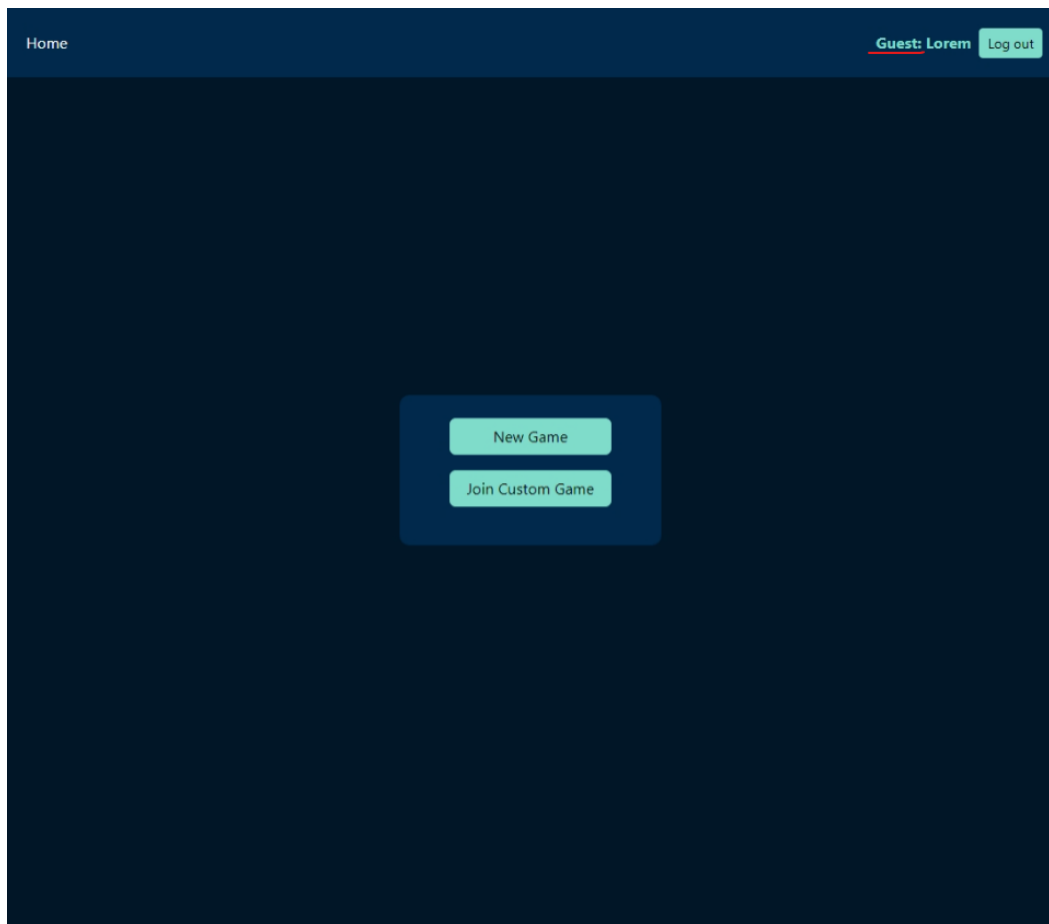
ábra 4.2:26 (tulajdonosi nézet)



ábra 4.2:27 (más játékosok nézete)

4.3 Vendégeknek

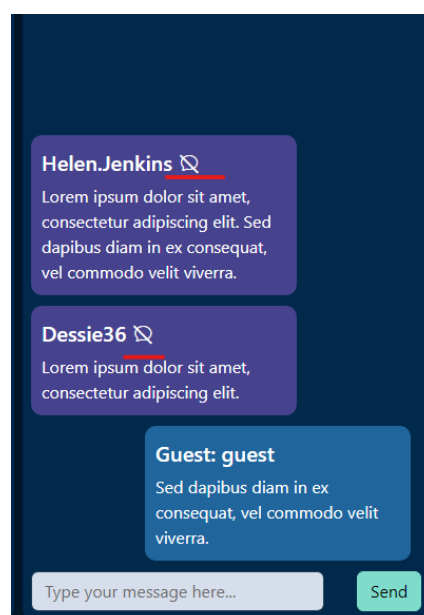
Vendég felhasználók korlátozottan férnek hozzá az oldalhoz és minden esetben a nevük előtt ott van a „Guest” jelző (ld. ábra 4.3:1).



ábra 4.3:1

Csoportos chat

Vendégek is hozzá férnek a játék és várószoba csoportos chat funkciójához, ám ehhez is csak korlátozottan. Némítani tudják csak a többi felhasználót, ezzel nem kapnak tőlük üzeneteket addig amíg nem oldják fel (ld. ábra 4.3:2).

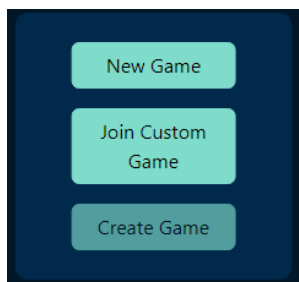


ábra 4.3:2

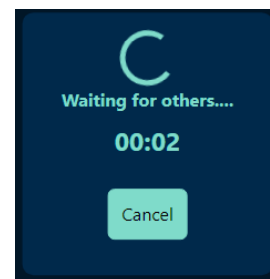
4.4 Játék (vendég- és regisztrált felhasználóknak)

A főoldalon megjelenő mini menüvel lehet új játékhoz csatlakozni, erre két lehetőség van:

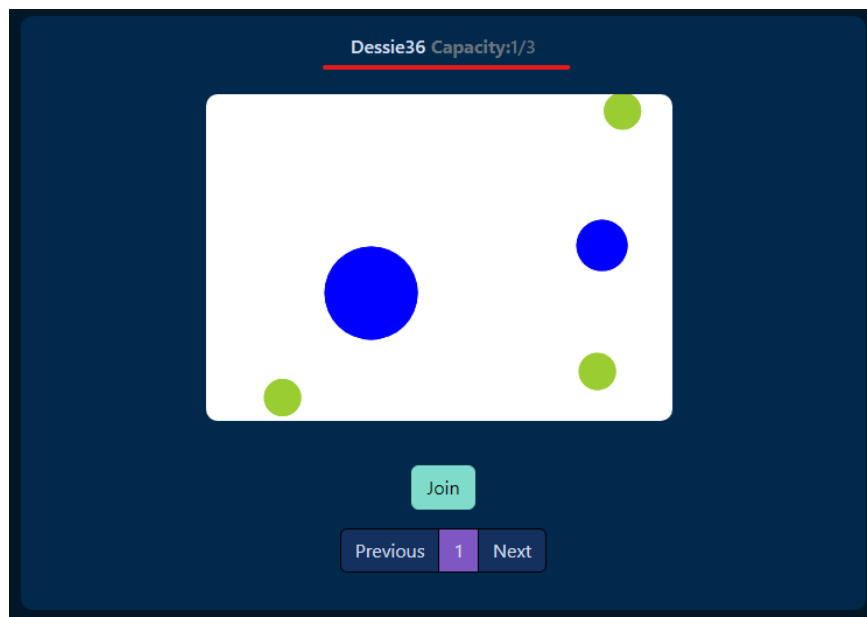
1. „New Game” gombra kattintva (ld. ábra 4.4:1): A játékos csatlakozik egy várólistára, abban az esetben, ha elég játékos van akkor elindítja a játékot. Amíg a játékosok a várólistán vannak, addig egy számláló jelenik meg, hogy mióta várnak (ld. ábra 4.4:2). A „Cancel” gombra kattintva ki lehet lépni a várólistáról.
2. „Join Custom Game” (ld. ábra 4.4:1): Egy lapozható lista jelenik meg, ahol a felhasználók által létrehozott játékok vannak és a pályák kicsinyítve láthatóak (ld. ábra 4.4:3). Továbbá az is látható, hogy ki készítette, és mennyien vannak bent, és mekkora a kapacitás.



ábra 4.4:1

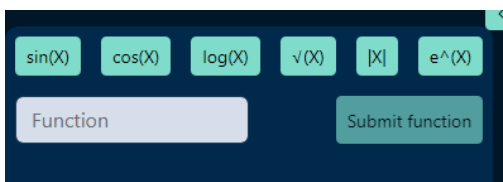


ábra 4.4:2

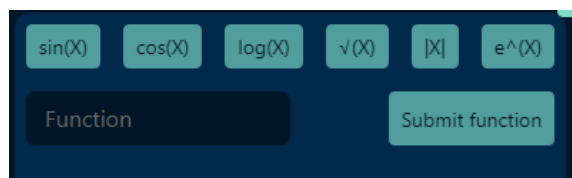


ábra 4.4:3

A játékhoz, mind a vendégek, mind felhasználók ugyan úgy férnek hozzá. Egy bemeneti mezőben meg lehet adni a függvényt, és a bemeneti mező felett láthatóak a használható függvények (ld. ábra 4.4:4). Ezekre rákattintva az oldal automatikusan beilleszti az adott függvényt a kurzor helyére. Azoknak a játékosoknak, akik éppen nincsenek soron azoknak az input mező zárolva van, és fekete háttérrel jelenik meg (ld. ábra 4.4:5).



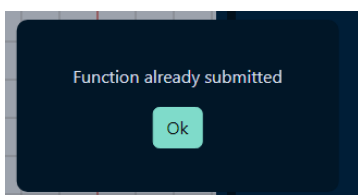
ábra 4.4:4



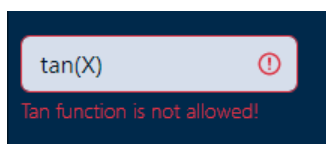
ábra 4.4:5

A függvények több szempont alapján is ellenőrizve vannak, ezek a következők:

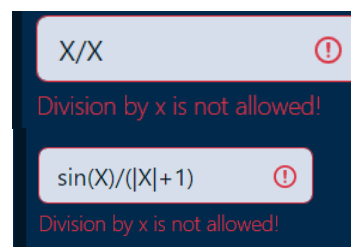
1. „Spam”/Használta-e már az adott játékos, tehát például egynél többször nem használható a $\sin(x)$ (ld. ábra 4.4:6).
2. Tangenst függvényt tartalmaz-e (ld. ábra 4.4:7).
3. $\frac{a}{x}$ alapú törtet tartalmaz-e pl.: $\frac{1}{x}$ vagy $\frac{x}{x}$ – utóbbi esetben ugyan konstans a függvény, de az oldal csak azt nézi, hogy a nevező tartalmaz-e x -t – (ld. ábra 4.4:8).
4. Az $a(x)$ vagy $x_1(x_2)$ — x_1 és x_2 bármilyen függvényt jelöl —, tehát a szorzás jelet minden esetben ki kell írni, kivételt képez az ax eset, tehát pl. a $2x$ -t elfogadja az oldal (ld. ábra 4.4:9).
5. A függvény érintkezik-e a bázissal, pl. Egy $|x| + 5$ függvény nem fog érintkezni a bázissal (ld. ábra 4.4:10).
6. Van-e olyan pont az intervallumon, melyen nem értelmezett a függvény — általában szakadás pontot jelent. Ez az eset alapvetően ki van szűrve korábbi validációkkal, de az az eset is ide tartozik, ha valami elírás szerepel pl. $\sin(x)$ helyett $son(x)$ van begépelve (ld. ábra 4.4:11).



ábra 4.4:6



ábra 4.4:7



ábra 4.4:8



ábra 4.4:9



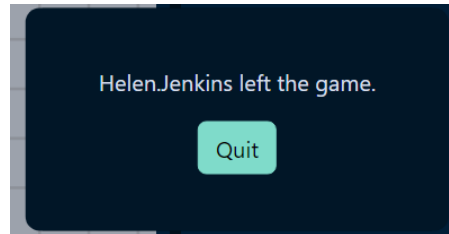
ábra 4.4:10



ábra 4.4:11

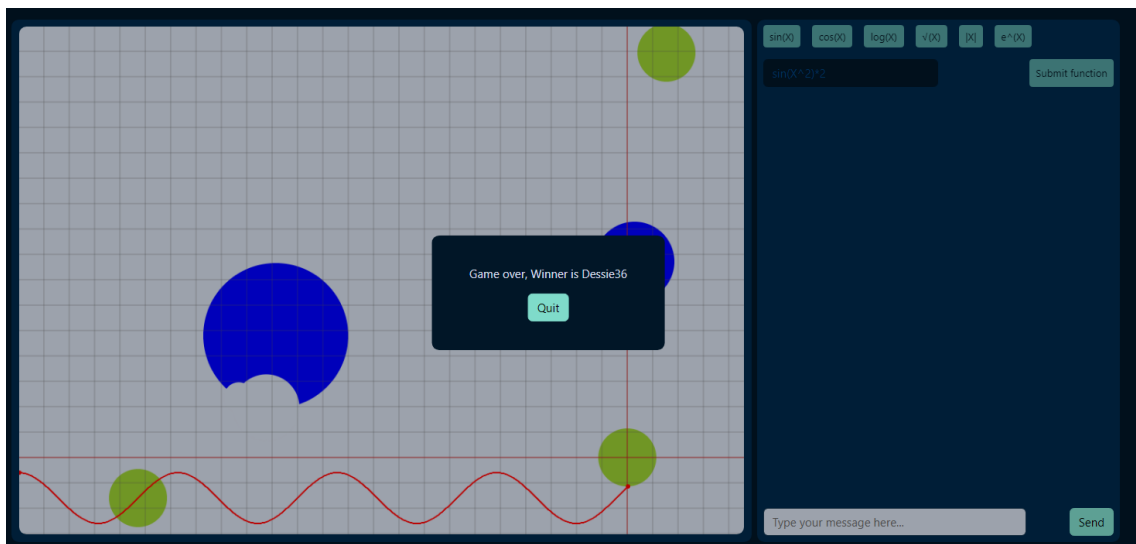
Játék közben előfordulhat, hogy valaki kilép. Ez kétféle módon lehetséges:

1. Elnavigál máshova az oldalon (pl. vissza a főoldalra). Ebben az esetben azonnal kilép a játékos, és a játék véget ér, melyről minden játékos kap egy üzenetet (ld. ábra 4.4:12).
2. Elmegy az internetje/véletlen bezárja az oldalt. Ilyenkor 10 másodperc áll rendelkezésére vissza csatlakozni a felhasználónak mielőtt a szerver véget vet a játéknak.



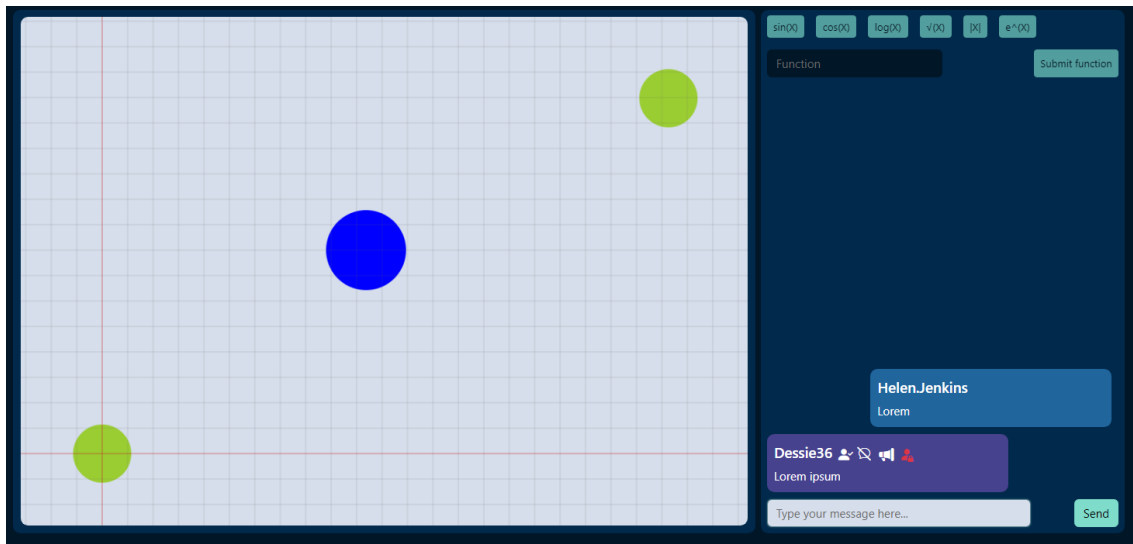
ábra 4.4:12

A játék akkor ér véget, ha valaki eltalálta másnak a bázisát, tehát 3 vagy 4 játékos esetén is elég, ha egy valakit eltaláltak. Ilyenkor felugrik egy ablak, hogy az „xy” játékos nyert (ld. ábra 4.4:13).

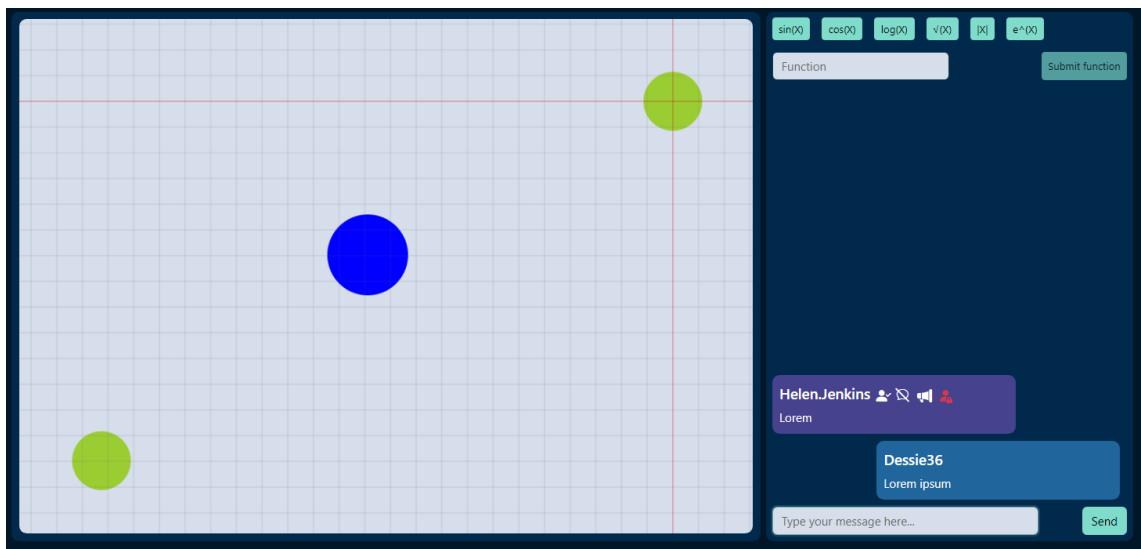


ábra 4.4:13

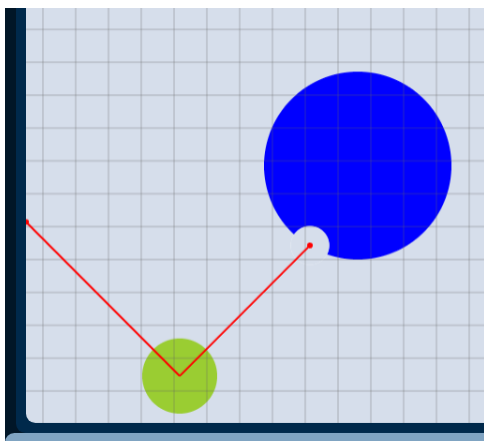
A pályán minden játékos a saját bázisának középpontjától látja az X és Y tengelyt egy piros vonallal (ld. ábra 4.4:14 és ábra 4.4:15). A pályán az akadályokat képesek elpusztítani, így biztosítva, hogy egy idő után mindenképp vége legyen a játéknak. A pusztításnak van egy minimális mértéke, mely annak függvényében növekszik, hogy a függvény mekkora utat tett meg addig (ld. ábra 4.4:16 és ábra 4.4:17).



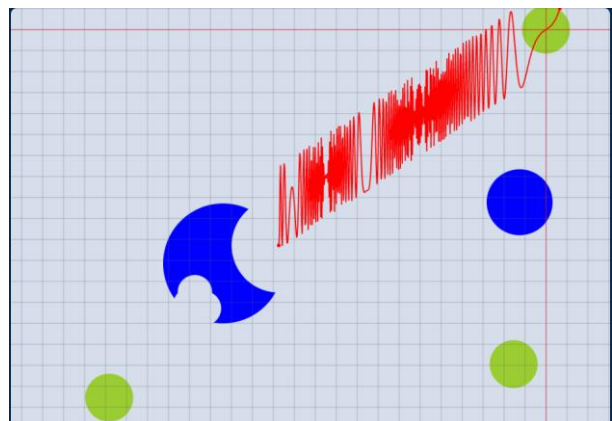
ábra 4.4:14



ábra 4.4:15



ábra 4.4:16



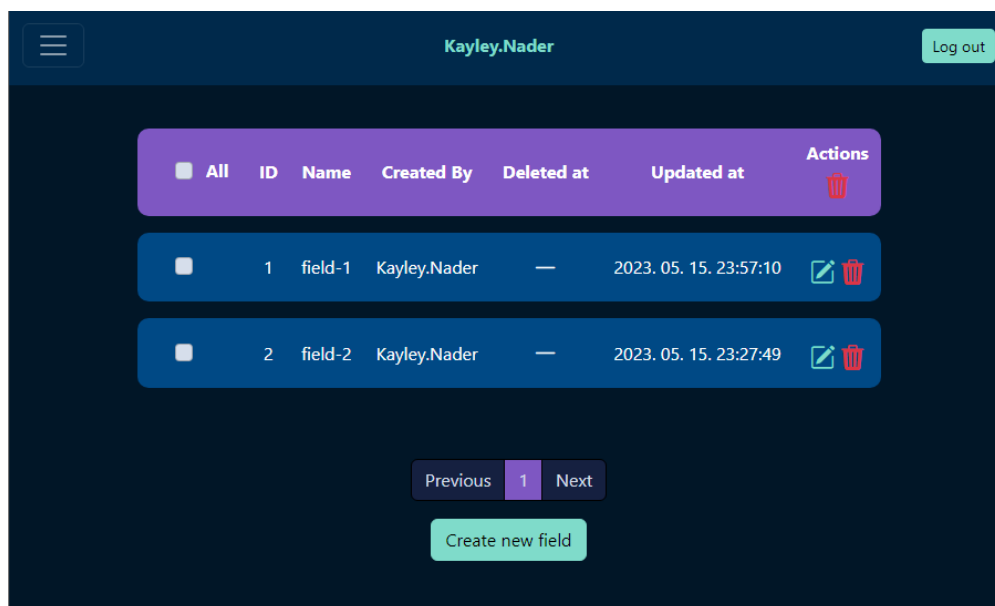
ábra 4.4:17





4.5 Pálya lista, pálya szerkesztés (adminoknak és felhasználóknak)

Pályalista

Az adminok a pálya listában minden admin felhasználónak a pályáját látják, és ezeket tudják szerkeszteni, ideiglenesen és véglegesen törölni, illetve helyre állítani (ld. ábra 4.5:1). A felhasználók csak a saját pályáikat tudják szerkeszteni, törölni és helyre állítani.

Minden pályához az ID, „Name”, „Deleted At”, „Updated At” és az „Actions” oszlop látható (ld. ábra 4.5:2), továbbá egy jelölő négyzet (ld. ábra 4.5:3) mellyel egyszerre több pályán is el lehet végezni ugyanazt a műveletet. Ahhoz, hogy egyszerre több adaton végezzék el a törlés műveletet, a fejléc sorban kell a törlés ikonra kattintani. Az adminok számára van egy plusz oszlop „Created By” (ld. ábra 4.5:1). A „Deleted At” akkor vesz fel értéket, ha törölve lett, és amennyiben van értéke, és ha megint a törlés művelet van rá alkalmazva, akkor véglegesen törlésre kerül. Ezen felül, ha törölve lett már akkor az „Actions” oszlopban megjelenik egy új ikon, amire klikkelve helyre lehet állítani (ld. ábra 4.5:5). A kis papír és ceruza ikont kiválasztva pedig a pályát lehet megnyitni szerkesztésre (ld. ábra 4.5:4). A lista alatt található egy gomb, melyre kattintva egy új pályát lehet létrehozni (ld. ábra 4.5:2).

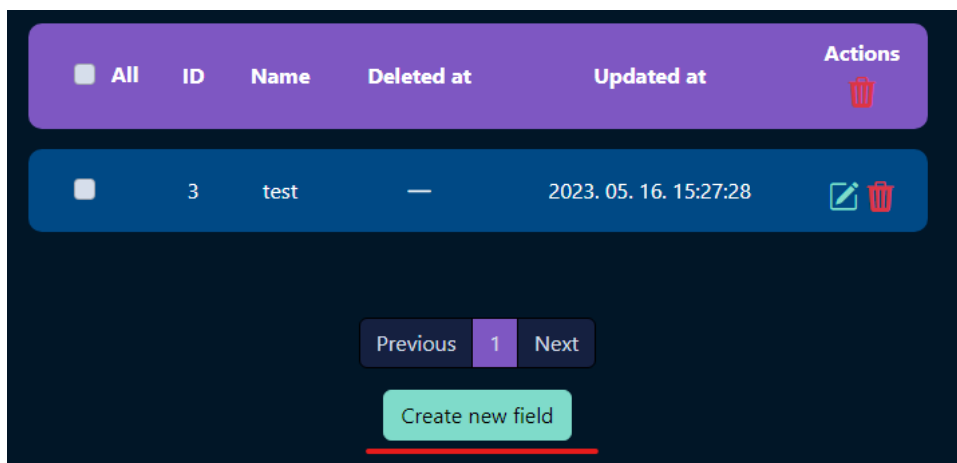


All	ID	Name	Created By	Deleted at	Updated at	Actions
<input type="checkbox"/>	1	field-1	Kayley.Nader	—	2023. 05. 15. 23:57:10	 
<input type="checkbox"/>	2	field-2	Kayley.Nader	—	2023. 05. 15. 23:27:49	 

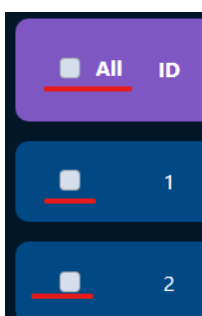
Previous 1 Next

Create new field

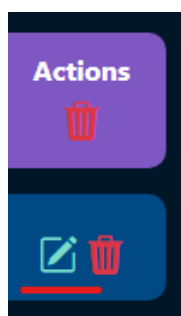
ábra 4.5:1 (admin nézet)



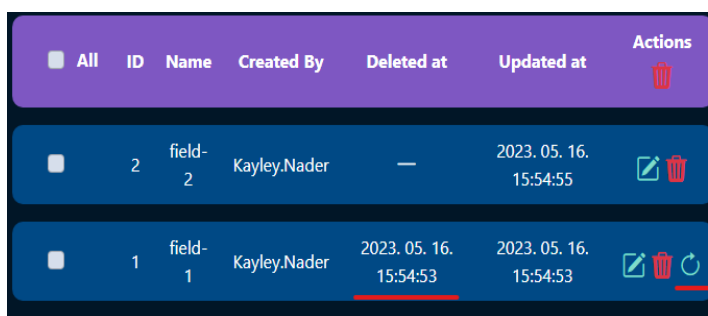
ábra 4.5:2 (felhasználói nézet)



ábra 4.5:3



ábra 4.5:4



ábra 4.5:5

Pályaszerkesztő

Miután egy admin vagy felhasználó a szerkesztés ikont, vagy a pálya létrehozás gombot választja, a pálya szerkesztő oldalra kerül (ld. ábra 4.5:6). A pálya mellett jobb oldalon találhatóak a vezérlő elemek. Az akadályokat úgy lehet elhelyezni, hogy az adott alakzatra kattintunk, ezzel az alakzat a pályára kerül (ld. ábra 4.5:7). Bázis helyezéséhez az „Add Player” gombra kell klikkelni.

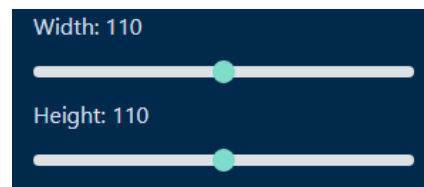
A kiválasztott elem mozgatható, ehhez a kívánt helyre kell kattintani, vagy az egér gombot nyomva tartva mozgatni. A kiválasztott akadálynak vagy játékos bázisnak a színe másrmilyen, és egyszerre csak egy lehet kiválasztva. A kijelölt akadály piros, míg a többi fekete és a kiválasztott bázis pedig neon zöld a többi pedig sötét zöld (ld. ábra 4.5:9). Egy akadályt vagy bázist úgy lehet kiválasztani, hogy rákattintunk. A kiválasztott akadályok méretét a „Width” és „Height” csúszkával lehet módosítani (ld. ábra 4.5:8 és ábra 4.5:10). A kiválasztott elem a „Remove selected” gombbal távolítható el. Az akadályok és a bázisok körül van egy halvány piros ellipszis (vagy kör), melynek szerepe, hogy az akadályok és a bázisok között elég nagy távolság legyen. Tehát, ha egy akadálynak a piros köre átfedésben van egy bázissal – vagy 2 bázis piros köre –, akkor hibát jelez a pálya szerkesztő (ld. ábra 4.5:11). A pályának nevet adni kötelező (ld. ábra 4.5:12).



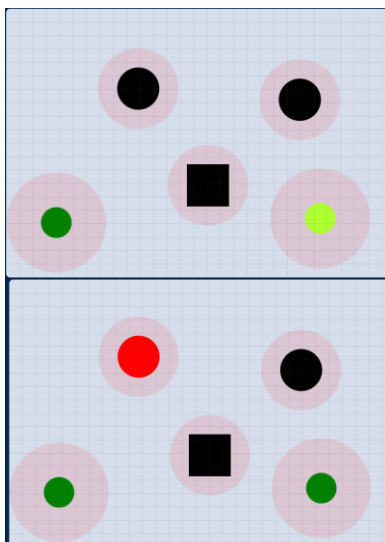
ábra 4.5:6



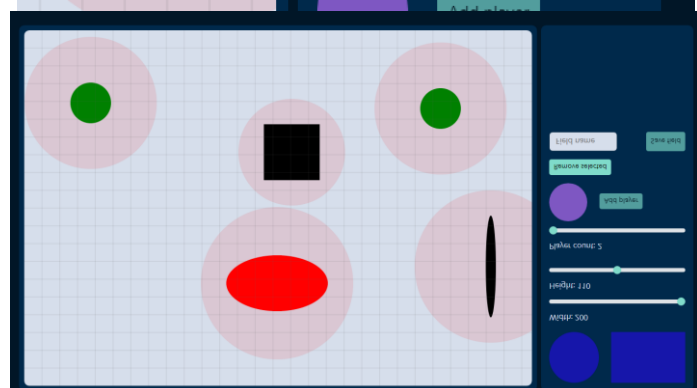
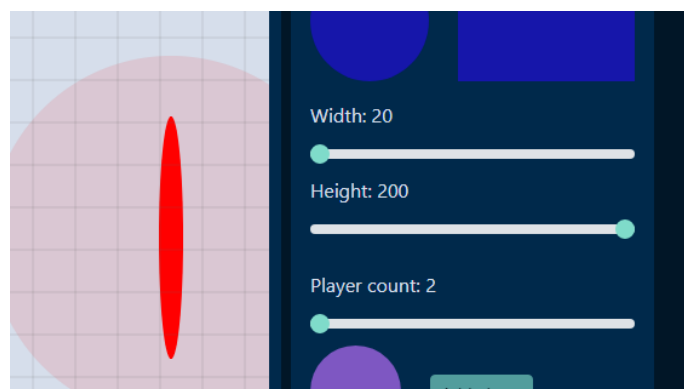
ábra 4.5:7



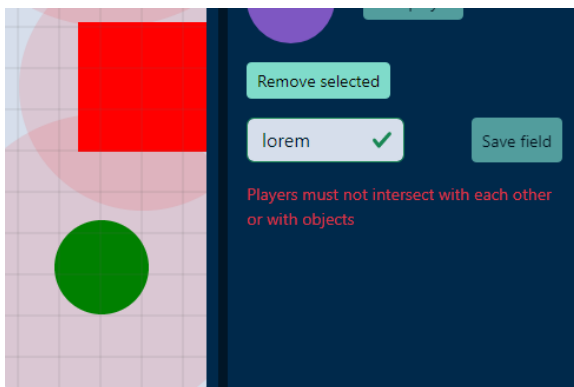
ábra 4.5:8



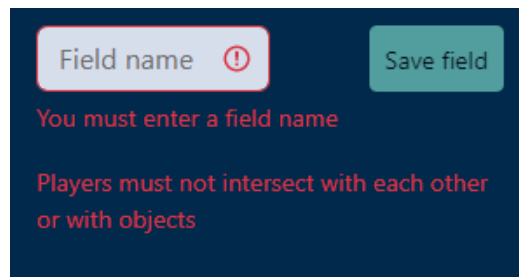
ábra 4.5:9



ábra 4.5:10



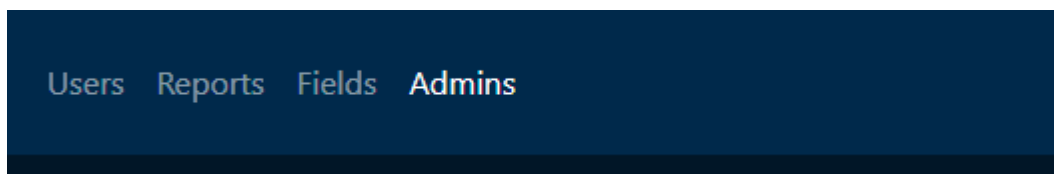
ábra 4.5:11



ábra 4.5:12

4.6 Adminoknak és a szuper adminnak

Az adminok és a szuper admin pályák létrehozásáért – ahol nincs kiemelve a szuper admin ott az adminok táborába tartozik a továbbiakban –, a felhasználók, az adminok és a jelentések kezelésért felelnek. Ezekre külön aloldalak vannak, illetve a főoldalon a felhasználók listája fogadja az adminokat (ld. ábra 4.6:1). Az oldalak között a navigációs sávban lehet váltogatni.



ábra 4.6:1

Listák általánosságban egy jelölő négyzetet tartalmaznak az első oszlopban, a fejlécben van lehetőség az összes kipipálására. Az így kijelölt sorokra lehet egyszerre elvégezni ugyanazt a műveletet. Ezek a műveletek a fejléc utolsó oszlopában találhatóak. A 2. oszlopban minden esetben van egy ID elem, a többi oszlop tartalma változó attól függően, hogy milyen lista van. Az utolsó oszlopban pedig a műveletek láthatóak az adott sorhoz kapcsolódóan, ilyen pl. a törlés és szerkesztés. Ha a műveletek fölé viszi az egeret egy admin, akkor egy „tooltip” lesz látható. Minden lista alatt egy lapozó található, amellyel előre-hátra lehet lapozni, illetve adott oldalra is rá lehet kattintani. A lapozó első és utolsó oldal fixen látható, ezen felül még 5 további oldal is.

Minden listában 3 féle adat típus lehet, szöveg dátum és logikai, illetve egy plusz, ami azt jelöli, hogy nincs adat. Egy logikai mező lehet egy kis piros kör, amelyben X jel található, vagy egy zöld kör, melyben pedig egy pipa. Az olyan adatoknál, amiknek nincs értéke azoknál egy vonal jel jelenik meg (ld. ábra 4.6:2).

All	ID	Name	Email	Role	Banned	Banned reason	Chat restricted	Actions
<input type="checkbox"/>	1	Dessie36	user1@functionWars.com	user		—		
<input type="checkbox"/>	2	Helen.Jenkins	user2@functionWars.com	user		—		
<input type="checkbox"/>	3	Alexandrine5	user3@functionWars.com	user		—		
<input type="checkbox"/>	4	Noel.Monahan50	user4@functionWars.com	user		—		
<input type="checkbox"/>	5	Darlene_Quigley86	user5@functionWars.com	user		—		
<input checked="" type="checkbox"/>	6	Stone.Labadie	user6@functionWars.com	user		—		
<input type="checkbox"/>	7	Adrianna.Baumbach12	user7@functionWars.com	user		—		

Previous 1 2 3 4 5 ... 7 Next

ábra 4.6:2

Felhasználók listában

A felhasználók listában a felhasználó következő adatai láthatóak (ld. ábra 4.6:2):

1. Neve („Name” – szöveg),
2. e-mail cím („E-mail” – szöveg),
3. szerepe („Role” – szöveg),
4. ki lett-e tiltva („Banned” – logikai)
5. tiltás oka („Banned reason” – szöveg)
6. chat hozzáférése korlátozva van-e („Chat restricted” – logikai)

Minden felhasználóhoz 3 művelet hajtható végre.

1. Tiltás/tiltás feloldása (): Tiltja a felhasználót abban az esetben, ha még nem lett, illetve feloldja, ha már lett. Ez a művelet végrehajtható egyszerre több felhasználóra is.
2. Chat korlátozás/chat korlátozás feloldása (): Tiltáshoz hasonlóan módosítja chat korlátozást. Ez a művelet is végrehajtható több felhasználóra egyszerre.
3. Kinevezés adminná ().


Jelentések listája











































Jelentések listában szereplő adatok (ld. ábra 4.6:3):

1. Kezelt („Handled” – logikai): Az adott jelentés lett-e már kezelve valamelyik admin felhasználó által.
2. Ki jelentette („Reported by” – szöveg): Melyik felhasználó adta le a jelentést.
3. Leírás („Description” – szöveg): Jelentés okának leírása.

4. Jelentett felhasználó („Reported user” – szöveg): Jelentett felhasználó.
5. Tiltott („Banned” – logikai): A jelentett felhasználóra vonatkozik, hogy ki lett-e már tiltva.
6. Tiltás oka („Banned reason” – szöveg).
7. Chat hozzáférése korlátozva van-e („Chat restricted” – logikai).
8. Törölve ekkor („Deleted at” – dátum): Mikor lett törölve a jelentés.

Jelentések listában elérhető műveletek:

1. Tiltás/tiltás feloldása (

<input type="checkbox"/> All	ID	Handled	Reported by	Description	Reported user	Banned	Banned reason	Chat restricted	Deleted at	Actions
<input type="checkbox"/>	1		Toy.Wiza	Incidunt consequuntur alias quo rem dolor velit laborum vitae.	Tyra54		—		—	  
<input type="checkbox"/>	2		Letitia97	Nemo libero modi excepturi beatae nam minus adipisci id nihil.	Eileen_Borer		—		—	  
<input type="checkbox"/>	3		Letitia97	Perferendis consequatur earum.	Denis74		—		—	  
<input type="checkbox"/>	4		Trenton34	Quae aspernatur hic animi ullam temporibus.	Jillian_Walker		—		—	  
<input type="checkbox"/>	5		Trenton34	Quae numquam dolore velit ipsa in quam.	Edmund.Beatty24		—		—	  
<input type="checkbox"/>	6		Hassie_Rath	Repellat nostrum rerum quod repellendus reprehenderit consequatur.	Elijah.Predovic		—		—	  
<input type="checkbox"/>	15		Kitty.Stiedemann	Iusto dolores mollitia quas rerum amet temporibus nesciunt quidem incidunt.	Letitia97		—		—	  

ábra 4.6:3

Adminok listája

Adminok listában szereplő adatok (ld. ábra 4.6:4):

1. Név („Name” – szöveg).
2. E-mail cím („Email” – szöveg).
3. Szerep („Role” – szöveg): „admin” vagy „szuper_admin” lehet

Adminok listában csak egy művelet érhető el: az admin jog eltávolítása (👤). Erre minden admin képes, és csak a szuper adminról nem vehető le, továbbá az adminok saját magukról nem tudják levenni. A művelet egyszerre csak egy felhasználóra hajtható végre (ld. ábra 4.6:4).

All	ID	Name	Email	Role	Actions
<input type="checkbox"/>	37	Karianne.Von	admin0@functionWars.com	admin	
<input type="checkbox"/>	38	Freida3	admin1@functionWars.com	admin	👤
<input type="checkbox"/>	39	Anthony.Cormier	admin2@functionWars.com	admin	👤
<input type="checkbox"/>	40	Super Admin	super.admin@functionWars.com	super_admin	

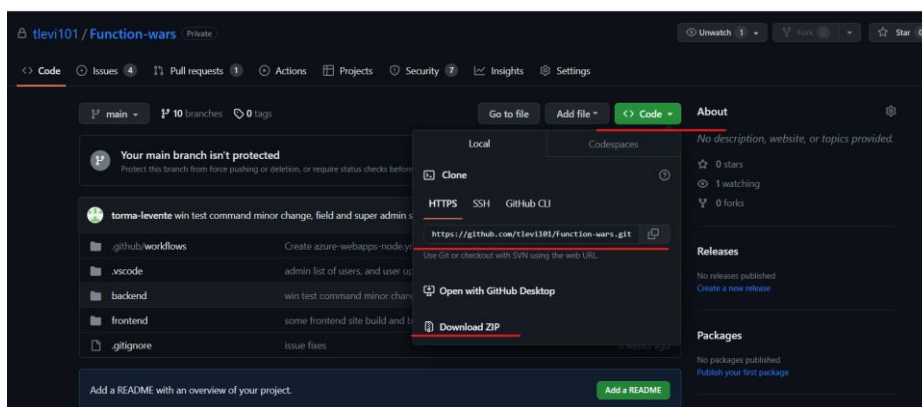
Previous 1 Next

ábra 4.6:4

4.7 Szuper adminoknak

Ebben a fejezetben a program telepítéséről lesz szó lokálisan. A program elérhető git használatával is, ehhez a [tlevi101/Function-wars \(github.com\)](https://github.com/tlevi101/Function-wars) oldalra kell menni, ott „Code” gombra kattintani, majd két lehetőség van (ld. ábra 4.7:1):

1. Klónozni: A „git clone <https://github.com/tlevi101/Function-wars.git>” parancsot kell futtatni (a linket a „clone” opció alatt HTTPS almenü pontban lehet elérni, a másolás ikonnal (📄) vágólapra lehet helyezni ld. ábra 4.7:1). Amennyiben még nincs „git” telepítve az eszközre, akkor linux esetén a „sudo apt-get install git” parancsot kell futtatni, windows esetén pedig el kell látogatni erre a linkre: [Git - Downloading Package \(git-scm.com\)](https://git-scm.com) és a „Standalone Installer” közül választani egyet a rendszernek megfelelőt (ld. ábra 4.7:2), majd lefuttatni.
2. ZIP formátumban letölteni: Ehhez nem kell semmit pluszban telepíteni, csak ki kell csomagolni.



ábra 4.7:1

Standalone Installer
32-bit Git for Windows Setup.
64-bit Git for Windows Setup.

ábra 4.7:2

Majd kellene fog egy adatbázis, erre két lehetőség van, egy MySQL adatbázis. MySQL telepítéséhez a következő lépéseket kell végre hajtani Windows rendszeren:

1. A MySQL hivatalos weboldalára kell menni és letölteni a „MySQL Community Server”-t: <https://dev.mysql.com/downloads/mysql/>
2. Ki kell választani a Windows operációs rendszert, majd a „Go to Download Page” gombra kattintani (ld. ábra 4.7:3).
3. A letöltött telepítőt el kell indítani.
4. A telepítés típusánál a „Developer Default” opciót kell választani (ld. ábra 4.7:5). Ezután a „Next” gombra kattintani.
 - a. A telepítéshez szükséges lehet a python telepítése is. Ehhez a python hivatalos oldalára kell látogatni, [Download Python | Python.org](#). És a „Download Python” gombra kattintani (ld. ábra 4.7:7). Majd a letöltött telepítőt elindítani.
5. „Installation” menüben az „Execute” gombra kell klikkelni, hogy elinduljon a szükséges programok telepítése (ld. ábra 4.7:6). Ha végzet, akkor a „Next” gombra kell kattintani.
6. A „Product Configuration” menüben a „Next”-re kell klikkelni, ez új menü pontokat nyit meg(ld. ábra 4.7:8).
7. A "Accounts and Roles" lépésben létre kell hozni a root felhasználót, ehhez egy jelszót kell megadni (ld. ábra 4.7:9). Jelszó megadása után lehet a „Next” gombra kattintani.
8. Az „Apply Configuration” menüig „Next”-re lehet klikkelni, majd az „Execute” gombot választani. Miután az „Execute” lefutott, lehet a „Finish” gombot megnyomni, ez visszavisz a „Product Configuration” menübe.
9. Ezt követően a „Next”-re kell kattintani megint, itt nincs teendő, tehát megint csak végig kell menni. Ezután ismét a „Product Configuration” menübe lép vissza, a „Next” gombra kell klikkelni.
10. A megadott jelszóval tesztelni kell a kapcsolódást a szerverhez (ld. ábra 4.7:10) ezután lehet „Next”-re kattintani, és aztán „Execute”-ra, majd, ha lefutottak a műveletek akkor lehet a „Finish”-re.
11. A telepítés kész, már csak végig kell menni a befejező lépéseken.

General Availability (GA) Releases
Archives

MySQL Community Server 8.0.33

Select Operating System:

Microsoft Windows

Looking for previous GA versions?

Recommended Download:

MySQL Installer for Windows

All MySQL Products. For All Windows Platforms. In One Package.

Starting with MySQL 5.6 the MySQL Installer package replaces the standalone MSI packages.

Windows (x86, 32 & 64-bit), MySQL Installer MSI

Go to Download Page >

Other Downloads:

ábra 4.7:3

MySQL Installer 8.0.33

Select Operating System:

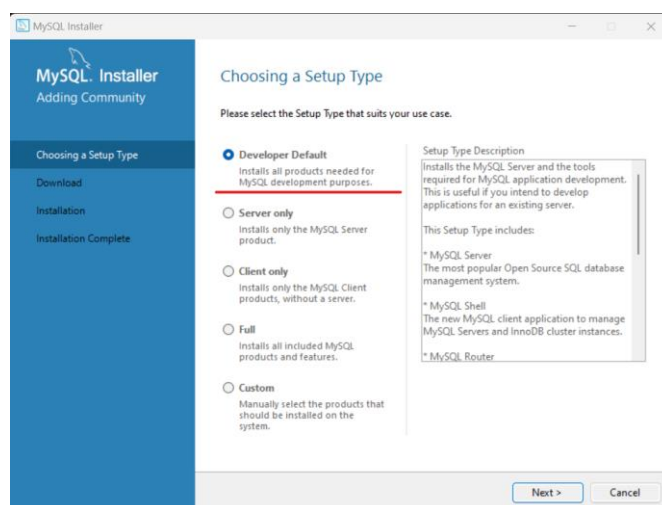
Microsoft Windows

Looking for previous GA versions?

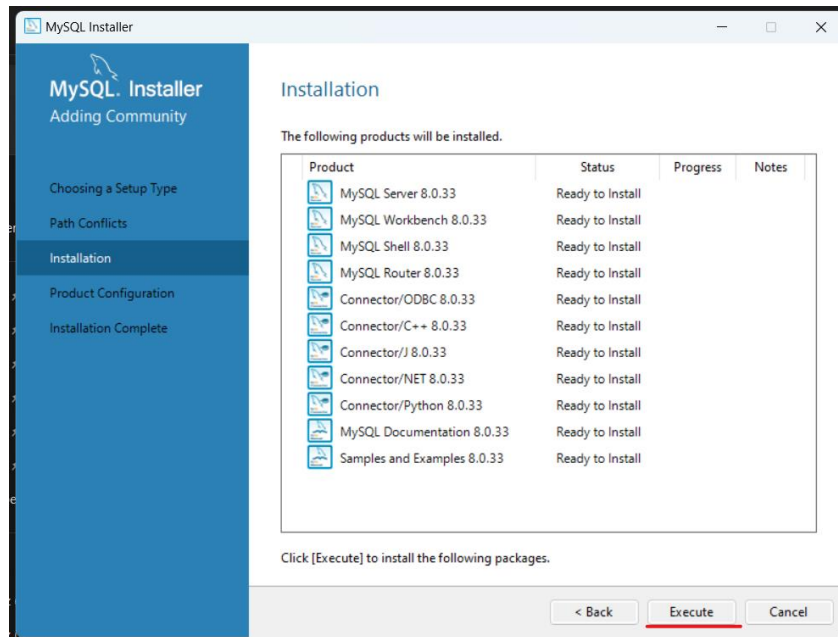
Windows (x86, 32-bit), MSI Installer	8.0.33	2.4M	Download
(mysql-installer-web-community-8.0.33.0.msi)			
MD5: 2a330cf24915964cca87e04dbb34e5d3 Signature			
Windows (x86, 32-bit), MSI Installer	8.0.33	428.3M	Download
(mysql-installer-community-8.0.33.0.msi)			
MD5: 9b4ce33ab05ae7e0aa30a6c4f1a4d1c2 Signature			

! We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

ábra 4.7:4



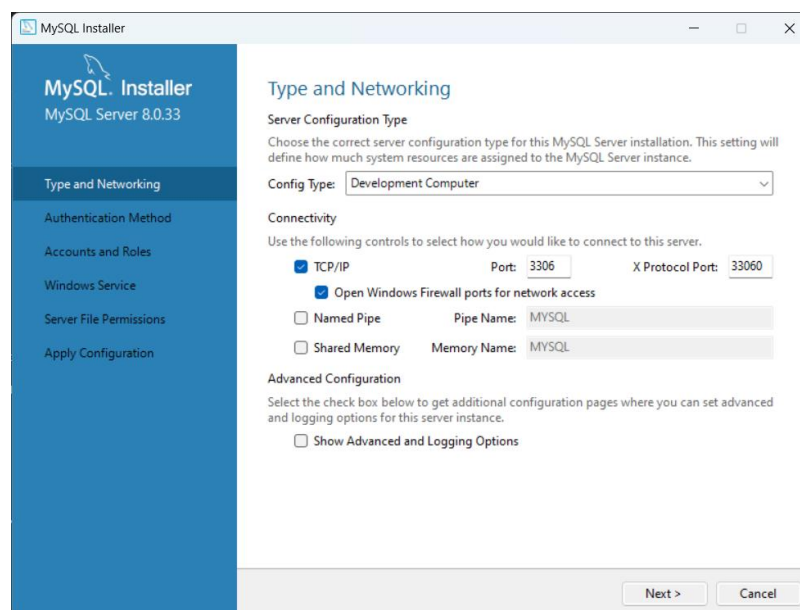
ábra 4.7:5



ábra 4.7:6



ábra 4.7:7



ábra 4.7:8

Accounts and Roles

Root Account Password

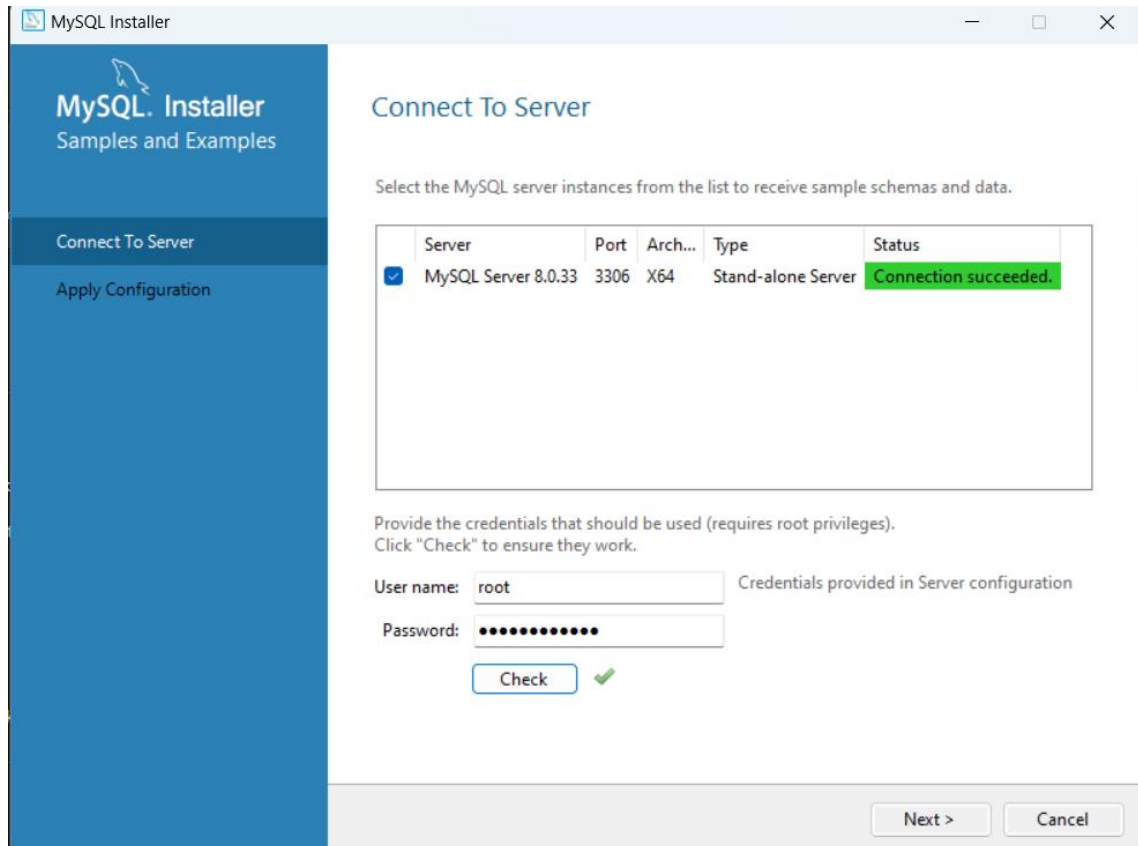
Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password:



Repeat Password:

ábra 4.7:9



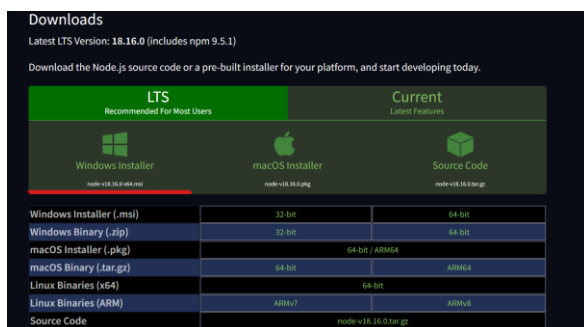
ábra 4.7:10

MySQL telepítése Linux rendszeren:

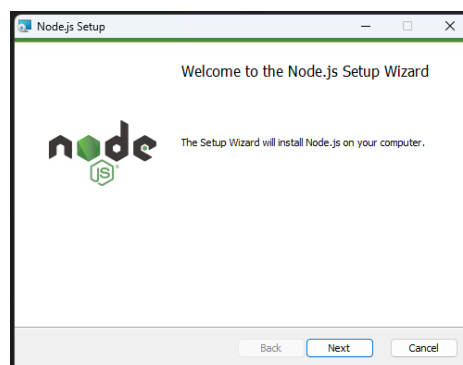
1. „sudo apt update”
2. „sudo apt install mysql-server”
3. „sudo systemctl start mysql.service”
4. Szerver beállítása:
 - a. „sudo mysql”
 - b. Majd a konzolba a következőt kell beírni: ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'new_password'; A new_password helyére meg kell adni a kívánt jelszót.
 - c. „Exit” – Ezzel a paranccsal ki kell lépni.
5. Kész, szerverhez lehet is csatlakozni.

Ha az adatbázis megvan, akkor a következő lépés a NodeJS telepítése. Ennek menete Windows rendszeren:

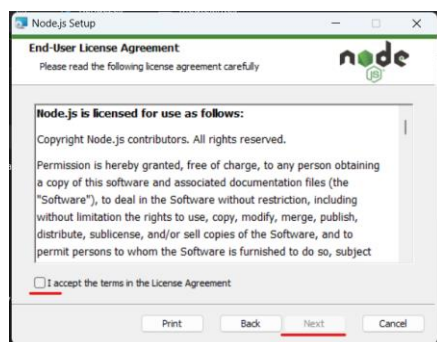
1. Látogasson el a [NodeJS hivatalos oldalára](#) majd válassza a „Windows Installer” letöltése opciót (ld. ábra 4.7:11).
2. Indítsa el a letöltött telepítőt.
3. Telepítés lépési az ábra 4.7:12-től az ábra 4.7:18-ig látható. Minden esetben a telepítő által kijelölt gombokra elég kattintani. Az „ábra 4.7:13”-as lépés során megjelenő szerződési feltételeket el kell fogadni.



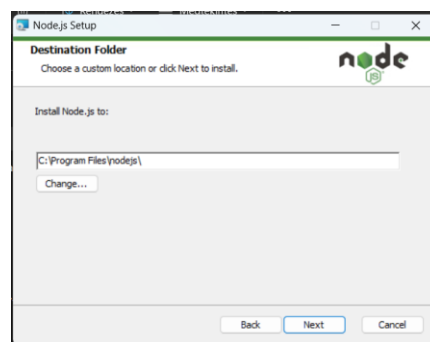
ábra 4.7:11



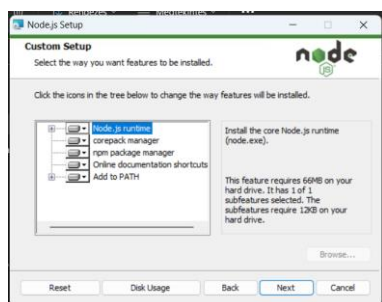
ábra 4.7:12



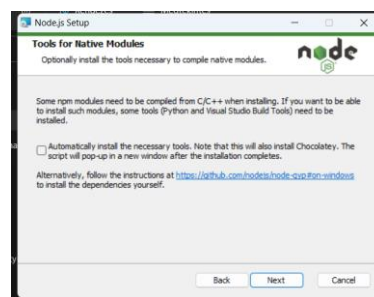
ábra 4.7:13



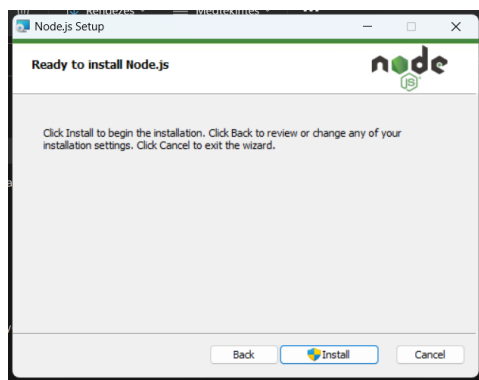
ábra 4.7:14



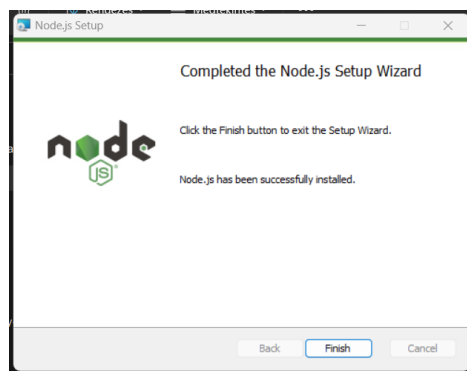
ábra 4.7:15



ábra 4.7:16



ábra 4.7:17



ábra 4.7:18

NodeJS telepítése Linux (Ubuntu) rendszeren:

1. „curl -fsSL https://deb.nodesource.com/setup_11.x | sudo -E bash - &&”
2. „sudo apt-get install -y nodejs”
3. Ellenőrzés: „node -v”, ha ez a parancs valamilyen verzió számot megjelenített akkor sikeres volt.

Ezután be kell lépni a „backend” mappába, ott található egy .env.example fájl, ezt át kell nevezni .env-re és meg adni a szükséges környezeti változókat (ld. ábra 4.7:19). A „DB_PASSWORD”-hoz be kell írni a feltelepített adatbázishoz tartozó „root” jelszót. Az e-mail küldéséhez a konfiguráció kihagyható abban az esetben, ha lokálisan nem szükséges annak tesztelése, de a mailtrap.io-n lehet ingyen „hamis” SMTP szervert létrehozni, tehát létező e-mail címekre nem lehet e-mailt küldeni – egy SMTP szerverhez meg vannak adva az adatok előre.

Ha a „.env” konfigurálása sikerült, akkor meg kell nyitni a terminált a „backend” mappában és egy „npm install” parancsot futtatni. Majd az „npm run db” parancsot kell futtatni, mely létrehozza az adatbázis táblákat, és létrehozza a szuper admin felhasználót. A szuper admin adatai a terminálban láthatóak (ld. ábra 4.7:20). Ezután már csak futtatni kell a szerveret, ehhez az „npm run dev” parancsot kell beírni.

```

1 *****
2 * SEQUELIZE *
3 *****
4 NODE_ENV=development # development, prod, test
5
6 *****
7 * JWT *
8 *****
9
10 JWT_SECRET="secret" # Change this to a random string on production
11 JWT_ALGO="HS256"
12
13 *****
14 * MYSQL DATABASE *
15 *****
16 DB_HOST=localhost
17 DB_PORT=3306
18 DB_USERNAME=root
19 DB_PASSWORD=
20 DB_DATABASE=function_wars
21
22 *****
23 * MAIL *
24 *****
25 # Your smtp server configuration
26 MAIL_NAME_FORGOT_PASSWORD=forgot.password@functionWars.com
27 MAIL_HOST=sandbox.smtp.mailtrap.io
28 MAIL_PORT=2525
29 MAIL_SECURE=
30 MAIL_USER=90449fa2470aad
31 MAIL_PASS=08216849a82b57

```

ábra 4.7:19

```

Sequelize CLI [Node: 18.16.0, CLI: 6.6.0, ORM: 6.31.0]

Loaded configuration file "config\dbConfig.js".
== 20230518110520-SuperAdminSeed: migrating =====
Super Admin created successfully!
Name: Super Admin
Please save this information in a safe place!
Email: super.admin@functionWars.com
Password: Qu8Joxb0
== 20230518110520-SuperAdminSeed: migrated (0.077s)

== 20230518111543-FieldSeed: migrating =====
== 20230518111543-FieldSeed: migrated (0.026s)

PS D:\Function-wars\backend> 

```

ábra 4.7:20

Mindezek után a „frontend” mappába kell lépni, majd terminálban „npm install”-t futtatni, és végül „npx ng serve” és a <http://localhost:4200/> oldalon el is érhető a projekt.

5 Fejlesztői dokumentáció

A program frontend¹ és backend² oldalra szét lett szedve teljes mértékben. A backend NodeJS API-ként funkcionál és a frontend API kéréseket küld. Frontenden Angular keretrendszert használok, és backenden pedig express³ és SocketIO⁴ szerver fut.

5.1 Backend

A backend oldalon vegyesen van JS és TS, eredetileg tisztán JS-nek indult, viszont ahogy nőtt a projekt szükségét láttam, hogy típus biztos legyen, és fordítási időben kiderüljenek a hibáim. Viszont a használt külső csomagok közül nem mindegyik támogatja a TS-et, így maradtam a vegyes megoldás mellett, de hogy egyértelmű legyen, hogy hol van TS, ezért létrehoztam a „types” mappát, melyben minden TS fájl megtalálható. Fordításkor minden általam létrehozott fájl a „dist” mappába kerül és onnan fut a szerver.

Backend megtervezésének elején sokat gondolkodtam azon, hogy PHP Laravel vagy NodeJS szerver legyen-e, és végül a NodeJS mellett döntöttem. Ennek fő oka, hogy játék működéséhez kellett websocketeket használni és NodeJS szerverhez már ismertem erre módot. Továbbá a függvényeket, mint frontend, mint backend oldalon ellenőrizni kell és ebben nagy segítség az, ha elég egyszer implementálni. Arra is lett volna lehetőségem, hogy backend oldalon ellenőrzöm, viszont ebben az esetben a frontend nagyon sok HTTP kérést küldött volna, és nem lett volna ugyan olyan jó a hiba vissza jelzés mintha frontend magától validálja. Ezért úgy gondolom, hogy az én megoldásom így a legjobb.

A socketekhez kapcsolódóan kell olyan adatokat tárolni, amik csak ideiglenesek, mint például „kik vannak fent jelenleg” stb. Az ilyen adatokat egy osztályban tárolom, minden ilyen adat statikus. Legtöbb adatszerkezet egy Map⁵-et valósít meg (ld. ábra 5.1:1).

```
1 export class RuntimeMaps {
2   public static games = new Map<string, Game>();
3   public static groupChats = new Map<string, GroupChat>();
4   public static waitList = new Map<number | string, socket>();
5   public static onlineUsers = new Map<number | string, { user: DecodedToken; socketID: string; currentURL: string }>();
6   public static waitingRooms = new Map<string, WaitingRoom>();
7   public static invites = new Set<{ inviterID: number; invitedID: number }>();
8   constructor() {}
9 }
```

ábra 5.1:1

Adatbázis

Táblák (ld. ábra 5.1:2)

1. Users

A felhasználók adatai találhatóak itt, a többi tábla is ehhez kapcsolódik. Itt kerül tárolásra, hogy az adott felhasználónak milyen jogai lettek megvonva.

¹ Kliensoldal

² Szerveroldal

³ Egy minimalista backend keretrendszer

⁴ Web socketek kezelésére szolgáló keretrendszer

⁵ Kulcs és érték párokat tartalmazó adatszerkezet

a. Mezők

- name: A felhasználó neve
- e-mail: A felhasználó e-mail címe
- password: A felhasználó jelszava (titkosítva)
- role: A felhasználó szerepe, „user” – sima felhasználó, „admin” – tiltani és chat hozzáférést nem lehet korlátozni, de el lehet venni admin jogait, ezzel újra csak „user” lesz. A „super_admin”-ra minden igaz, mint egy adminra, kivéve, hogy nem lehet elvenni az admin jogait, illetve szuper admint az oldalról nem lehet létrehozni. A szuper admin az adatbázis létrehozásával jön létre Sequelize seeder¹ segítségével.
- banned: A felhasználó tiltva van-e, ha igen, akkor nem fér hozzá az oldalhoz.
- chat_restriction: A felhasználónak a chat hozzáférése korlátozva van-e.
- createdAt: Létrehozás dátuma.
- updatedAt: Módosítás dátuma.

2. PasswordResets

A felhasználóknak lehetősége van jelszó visszaállításra, az ehhez szükséges adatok ebben a táblában kerülnek tárolásra.

a. Mezők

- user_id: Annak a felhasználónak az azonosítója, aki kérte a jelszó helyreállítást.
- link: Front-end linkje, tartalmazza „uuid”-t is.
- uuid: Teljesen egyedi id az adott jelszó helyre állításhoz.
- createdAt: Létrehozás dátuma.
- updatedAt: Módosítás dátuma.

3. Reports

A felhasználók jelenteni tudják egymást, ezen a táblán összekapcsolásra kerül két felhasználó, aki a jelentést tette, és aki jelentve lett.

a. Mezők

- reported_by: Annak a felhasználónak az id-ja, aki a jelentést adta.
- reported: Annak a felhasználónak az id-ja, aki jelentve lett.
- description: Jelentés oka/leírása, kötelező megadni.
- handled: Az adminok foglalkoztak-e vele.
- deletedAt: „soft delete” alkalmazására szolgál, ha ez null, akkor még nem lett törölve.
- createdAt: Létrehozás dátuma.
- updatedAt: Módosítás dátuma.

4. Fields

A felhasználók és az adminok által létrehozott pályák kerülnek tárolásra itt.

a. Mezők

- user_id: A létrehozó id-ja.
- admin_field: Admin hozta-e létre a pályát, ha igen, akkor minden admin tudja ezt módosítani.
- field: Pálya adatai JSON formátumban.
- deletedAt: „soft delete” alkalmazására szolgál, ha ez null, akkor még nem lett törölve.
- createdAt: Létrehozás dátuma.
- updatedAt: Módosítás dátuma.

5. Blocks

Egy adott játékos által tiltott játékosok szerepelnek itt.

a. Mezők

- user_id: Aki tiltotta a másik felhasználót.
- blocked_id: Aki tiltva lett.

¹ Feltölti az adatbázist adatokkal, ezek lehetnek véletlenszerű adatok, amik csak a fejlesztési fázist segítik, vagy végleges adatok a kész programhoz.

6. FriendShips

A felhasználók közötti baráti kapcsolatokat tárolja.

a. Mezők

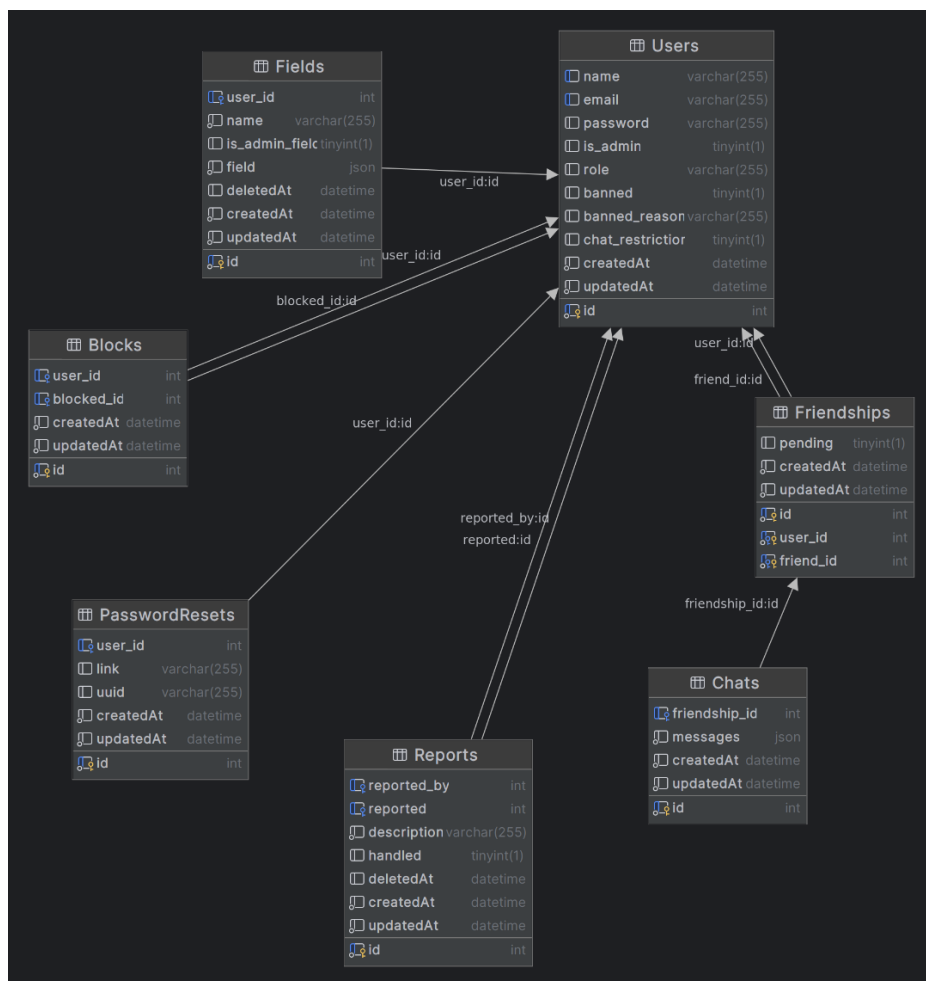
- user_id: Aki a barátságot kezdeményezte.
- friend_id: Aki a barátság kérelmet kapta.
- pending: A barátság kérelem állapota, ha „true” akkor még nem barátok, ha „false” akkor már barátok.
- createdAt: Létrehozás dátuma.
- updatedAt: Módosítás dátuma.

7. Chats

A barátok közötti beszélgetéseket tárolja.

a. Mezők

- friendship_id: Kikhez tartozik a chat.
- messages: Az üzenetek egy JSON mező, ami úgy néz ki, hogy van egy tömb, benne a következő objektumok: { from: number, message:string, seen: boolean }.



ábra 5.1:2 (Adatbázis táblák és kapcsolataik), Diagrammot a JetBrains DataGrip generálta

Sequelize seederek

Seederek feladata, hogy feltöltsék az adatbázist adatokkal. Háromféle seeder csoportot definiáltam, dev – fejlesztői környezet, test – tesztesetekhez és prod (production) – felhasználók által használt végtermékhez.

Development seeder random adatokkal tölti fel az adatbázist, ehhez a „faker” csomagot használtam, amivel szinte bármilyen hamis adatot le lehet generálni, nevek, e-mail címek, jelszavak, cég nevek, felhasználó nevek, mondatok, nagyobb szövegek – utóbbiak random „lorem ipsum” szövegek – és sok egyéb más is generálható vele. Fejlesztői környezethez a seeder 30 és 50 között hoz létre felhasználókat, nevüket a fakerrel generálom (`faker.internet.userName()`), az e-mail cím viszont egységesen ``user${index+1}@functionWars.com`` ehhez hasonlóan a jelszó is egységes „password”. Admin felhasználókból 2-3 lehet – plusz a szuper admin –, akik a rendes felhasználókhoz hasonlóan vannak létrehozva, név random generált és az e-mail cím ``admin${index+1}@functionWars.com``, jelszavuk ugyan úgy „password”. Szuper admin már kicsit másképp működik, neve „Super Admin” e-mail címe super.admin@functionWars.com jelszava viszont random generált (`faker.internet.password(len: 8)`), és seeder futásának végén ezek ki vannak írva (ld. ábra 4.7:20). Továbbá felhasználók közötti barátság kapcsolat is létrejön, minden felhasználónak 20-30 barátja lesz, ezek random vagy függőben lévő barátságok vagy már elfogadottak (`pending: faker.datatype.boolean()`). Minden felhasználóhoz létrejön 0-3 jelentés (report).

Production seederben létrejön a szuper admin, egy 2, 3 és 4 fős pálya.

Test seeder hasonlóan működik, mint a development, csak létrejönnek extra fix adatok is, amik a teszteléshez kellenek.

Sequelize modellek

Sequelize szinte teljes mértékben átveszi az adatbázis kezelését, csak egy modellt és egy „migration”-t kell definiálni. Modell kezeli a táblákat, indít lekérdezéseket és létrehoz új adatokat. Egy „migration” csak azért felel, hogy a táblák létre legyenek hozva az adatbázisban.

SQL lekérdezések írása helyett a modellekkel elég csak függvény hívásokat intézni. Egy konkrét adat lekérése ID alapján úgy történik, hogy a modellre meg kell hívni a „findByPk” függvényt pl.: `User.findByPk(1)`. Ha valamilyen bonyolultabb feltételt szeretnénk megadni, akkor ezt a „findAll” függvénnyel tehetjük meg, és egy „where” feltételt kell megadni (ld. ábra 5.1:3).

```
1 Friendship.findAll({
2   where: {
3     friend_id: this.id,
4     pending: true,
5   }
6 });
```

ábra 5.1:3

1. User

Metódusok:

- `comparePassword(password: string)`

Összehasonlítja a felhasználó „hashel”-t jelszavát a paraméterben kapott jelszóval, és „true”-val tér vissza, ha egyeznek.

- toJSONForJWT()
Felhasználó adataival tér vissza, kivétel a jelszó. Illetve két plusz adatot is hozzá ad, JWT_createdAt: aktuális dátum és idő és type: „user”.
- toJSON()
Felhasználó adataival tér vissza, kivétel a jelszó.
- async¹ getFriends()
A barátok listájával tér vissza.
- async getFriendRequests()
Minden függőben lévő barátságkapcsolat („Friendship”).
- async getChat(friend_id:number)
Visszatér a baráthoz chathez.
- async getFriendShip(friend_id: number)
Visszatér egy Friendship modellel, ami az aktuális modell és a barát közötti barátságot írja le. Ha nincs barátság a felhasználók között akkor „null”-al tér vissza.
- async isBlocked(other_user_id:number)
„true”, ha a másik felhasználó tiltva van, „false”, ha nem.
- async isFriend(other_user_id:number)
„true”, ha a másik felhasználó barát van, „false”, ha nem.

Modell események²:

- beforeCreate:
Mielőtt a user létrejönne az adatbázisban, a jelszó titkosításra kerül („hash”).
- beforeUpdate:
Ha a nincs titkosítva a jelszó, az az módosult, akkor titkosításra kerül.

2. PasswordReset

Metódusok:

- toJSON()
A Passwordreset adataival tér vissza.
- getUuid()

Modell események:

- beforeCreate:
Létrehoz egy egyedi azonosítót (UUIDv4) és linket is létrehozza ez alapján („reset-password/a_generált_uuid”).

3. Field

¹ Aszinkron függvény, melynek lefutását be kell várni.

² Minden modellhez tartoznak események pl. beforeCreate, beforeUpdate

Metódusok:

- `static async randomField(playerLimit = 2)`¹
Visszatér egy random Field-el, ami 2 játékosra lett tervezve.

¹ Alapértelmezett paraméter a 2

Sequelize modellek kapcsolatai

A modellek közötti kapcsolatok definiálásához minden modell elején van egy statikus „associate” függvény, ahol definiálni lehet egy kapcsolatot (ld. ábra 5.1:4). A „foreignKey” azt jelöli, hogy a másik táblában melyik oszlop hivatkozik a modellhez tartozó tábla elsődleges kulcsára. Az „as”-el el lehet nevezni a kapcsolatot, illetve a sequelize ezt használja ahhoz, hogy a kapcsolatokat kezelje, azaz, hogy hozzá adjon a modellhez egy másikat, lekérje a modellhez kapcsolódó összes többi modellt. Például a „mySentReports” kapcsolatnál úgy lehet hozzáadni új jelentést („Report”), hogy a modellre meghívjuk az „addMySentReport”¹ függvényt. Lekérni pedig úgy lehet, hogy „getMySentReports”-ot hívjuk.

```
1 static associate(models) {  
2   this.hasMany(models.Report, {  
3     foreignKey: 'reported_by',  
4     as: 'mySentReports',  
5   });  
6 }
```

ábra 5.1:4

1. User – User: Report
Felhasználók között „Many-to-Many”² kapcsolat van a „Reports” kapcsolótáblával. Ezt a kapcsolatot a Report modell kezeli.
2. Users – PasswordReset
A két tábla között egy „One-to-One”³ kapcsolat áll fenn, minden felhasználónak csak egy jelszó helyreállító linkje lehet.
3. User – User: Friendship
Felhasználók között „Many-to-Many” kapcsolat van a „friendships” kapcsolótáblával. Ezt a kapcsolatot a Friendship modell kezeli, hogy a barátságoknál a függőben lévő barátságokat lehessen kezelni.
4. User – User
Felhasználók között a tiltásokat leíró „Many-to-Many” kapcsolat. Kapcsolótábla a „Blocks”
5. User – Field
A két tábla között egy „One-to-Many”⁴ kapcsolat áll fenn. Ez azt jelenti, hogy egy felhasználónak több pályája is lehet, de egy pálya csak egy felhasználóhoz tartozhat. Admin pálya esetén, az a kivétel, hogy az adminok látják és tudják szerkeszteni egymás pályáit.
6. Friendship – Chat
Ez egy „One-to-One” kapcsolat, egy „friendship”-nek csak egy chat-je lehet, ha a barátság felbomlik akkor a chat is törlődik.

¹ Nem elírás történt, ilyen esetben lekerül a többes szám, viszont lehet olyat is, hogy „addMySentReports” ilyenkor viszont a paraméter egy tömböt vár.

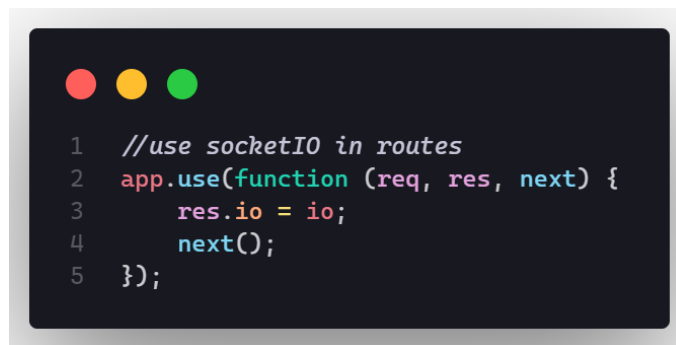
² A modell tartozhat több másik modellhez is és a modellhez tartozhat több másik modell is.

³ A modellhez pontosan egy másik modell tartozhat

⁴ A modellhez tartozhat több másik modell is.

API

Az API-hoz a „rout”-ok¹ a routes mappában vannak létre hozva, és „controller”-ek, amik pedig kezelik a kéréseket, melyek a „types/controllers” mappában találhatóak. Vannak API hívások, melyek socket eseményeket is indítanak, ehhez pedig az App.js fájlban hozzá adtam a „socketIO”-t (ld. ábra 5.1:5). Erre azért volt szükség, mert vannak bizonyos socket események, amiket validálni kell. Viszont arra socket nem képes, hogy hibát adjon vissza a kliensnek, ilyen pl. mikor egy játékos küld egy függvényt, a backend válaszolhat egy hibával.

A screenshot of a code editor with a dark background and light-colored text. The code is written in JavaScript and is used to configure Express.js routes to use Socket.io. It consists of five lines of code, numbered 1 to 5. The code uses the 'app.use' method to register a middleware function that sets 'res.io' to the 'io' object and then calls 'next()'.

```
1 //use socketIO in routes
2 app.use(function (req, res, next) {
3   res.io = io;
4   next();
5 });
```

ábra 5.1:5

API végpontok

- I. Autentikációs végpontok
 1. POST /register-guest
 2. POST /register
 3. POST /login
 4. POST /forgot-password
 5. PUT /reset-password/:UUID
 6. GET /update-token
- II. Admin műveletekhez kapcsolódó végpontok
 1. GET /admins
 2. GET /users
 3. GET /reports
 4. DELETE /reports/:id
 5. PUT /users/:id/ban
 6. PUT /users/:id/unban
 7. PUT /users/:userID/make-admin
 8. PUT /users/:userID/remove-admin
 9. PUT /users/:id/add-remove-chat-restriction
- III. Egyéni játékhoz kapcsolódó végpontok
 1. GET /custom-games
 2. GET /wait-rooms/:roomUUID
 3. POST /wait-rooms/:roomUUID/userID/kick
- IV. Pályákkal kapcsolatos végpontok
 1. GET /fields
 2. GET /field/:id
 3. GET /fields/:id/show

¹ API végpontok, amikre érkezik a kérés (request)

4. POST /fields
5. PUT /fields
6. DELETE /fields/:id
7. PUT /fields/:id/restore
- V. Játékkal kapcsolatos végpontok
 1. POST /:gameUUID/function
 2. GET /:gameUUID
- VI. Chat szóbával kapcsolatos végpontok
 1. POST /:roomUUID/mute/:userID
 2. POST /:roomUUID/unmute/:userID
 3. GET /:roomUUID/messages
 4. GET /:roomUUID/users-status
- VII. Barátokkal és más felhasználókkal kapcsolatos végpontok
 1. GET /friends
 2. GET /friends/online
 3. GET /friends/requests
 4. GET /friends/:id/chat
 5. PUT /friends/requests/:id/accept
 6. DELETE /friends/requests/:id/reject
 7. DELETE /friends/:id
 8. POST /friends/:id
 9. POST /users/:id/report
 10. POST /users/:id/block
 11. DELETE /users/:id/unblock

API dokumentáció

A hibaüzenet minden esetben egy JSON objektum, ami egy „message” elemet tartalmaz. A sikeres válaszok is egy JSON objektumot tartalmaznak, ezeknek az elemei az adott helyen vannak felsorolva.

- I. Autentikációs végpontokhoz (AuthController kezeli ezeket a végpontokat):
 1. POST /register-guest – Vendég felhasználók regisztrálását erre a végpontra kell küldeni.

Body¹:

 1. name (felhasználó által megadott név)

Követelmények:

 - a. kötelező
 - Státusz kód: 400 – „Name is required”
 - b. minimum 3 karakter és maximum 20 karakter
 - Státusz kód: 400 – „Name is must be between 3 and 20 characters long”

Siker esetén:

 2. Státusz kód: 201
 - a. message: „Guest accepted”
 - b. jwt (string) – dekódolás után egy „interface”-t elégít ki (ld. ábra 5.1:6)

¹ API hívás törzse

2. POST /register – új felhasználók regisztrálása

Body:

1. name
 - a. kötelező
 - Státusz kód: 400 – „notNull Violation: User.name cannot be null” *
 - b. minimum 3 karakter és maximum 20 karakter
 - Státusz kód: 400 – „Validation error: Username must be between 3 and 20 characters” *
 - c. egyedi
 - Státusz kód: 400 – „users_name must be unique” *
2. e-mail
 - a. kötelező
 - Státusz kód: 400 – „notNull Violation: User.e-mail cannot be null” *
 - b. e-mail formátumú
 - Státusz kód: 400 – „Validation error: Invalid e-mail address” *
 - c. egyedi
 - Státusz kód: 400 – „users_e-mail must be unique” *
3. password
 - a. kötelező
 - Státusz kód: 400 – „Password is required”
 - b. minimum 8 karakter és maximum 20 karakter
 - Státusz kód: 400 – „Validation error: Password must be between 8 and 20 characters” *
4. passwordAgain
 - a. kötelező
 - Státusz kód: 400 – „Password confirmation is required (passwordAgain)”
 - b. meg kell egyeznie a password mezővel
 - Státusz kód: 400 – „Passwords do not match”

* Másik hibaüzenet is tartozhat hozzá. A hibaüzenetek új sor karakterrel vannak elválasztva („\n”)

Siker esetén:

1. Státusz kód: 201
 - a. message: „User created”
 - b. jwt (string): dekódolás után egy „interface”-t elégít ki (ld. ábra 5.1:7)

3. POST /login – felhasználók beléptetése

Body:

1. E-mail
 - a. Kötelező
 - Státusz kód: 400 – „E-mail required”
 - b. Létezzen felhasználó a meg adott e-mail címhez.
 - Státusz kód: 404 – „Incorrect e-mail”

2. Password

a. Kötelező

- Státusz kód: 400 – „Password required”

b. Nem megfelelő e-mail

- Státusz kód: 400 – „Incorrect password”

Egyéb hibaüzenetek:

1. Státusz kód: 403

- a. { message: „User banned”, banned_reason: string }

Siker esetén:

1. Státusz kód: 200

- a. message: „Login successful”
- b. jwt: string – dekódolás után egy „interface”-t elégít ki (ld. ábra 5.1:7)

4. POST /forgot-password – jelszó helyreállító e-mail küldése

Body:

1. E-mail

a. Kötelező

- Státusz kód: 400 – „E-mail required”

b. Létezzen felhasználó a meg adott e-mail címhez.

- Státusz kód: 404 – „Incorrect e-mail”

Egyéb hiba üzenetek:

1. Státusz kód: 403

- a. { message: „User banned”, banned_reason: string }

Siker esetén:

1. Státusz kód: 201

- a. message: „E-mail sent”

5. PUT /reset-password/:UUID – jelszó helyreállítása

Paraméter:

1. UUID

a. Legyen ilyen link.

- Státusz kód: 404 – „Incorrect link”

b. Érvényes link – a linkek 1 óra után lejárnak

- Státusz kód: 400 – „Link expired”

Body:

1. password

a. kötelező

- Státusz kód: 400 – „Password required”

b. minimum 8 karakter és maximum 20 karakter

- Státusz kód: 400 – „Validation error: Password must be between 8 and 20 characters”

2. passwordAgain

a. kötelező

- Státusz kód: 400 – „Password confirmation is required (passwordAgain)”

b. meg kell egyeznie a password mezővel

- Státusz kód: 400 – „Passwords do not match”

6. GET /update-token – Azt a célt szolgálja, hogy valahányszor egy felhasználó betölti az oldalt, akkor egy friss JWT tokenet kapjon.

Header¹:

1. Authorization (autentikáció)

Típusa „Bearer” és tartalmaznia kell a JWT token

- a. Hibaüzenet:

- Státusz kód: 401 – „Unauthorized” (token hibás)

Hibaüzenetek:

1. Státusz kód: 403

- a. „Guest cannot make this request” – figyelni kell arra, hogy vendég felhasználók esetében ne legyen ilyen küldve.

2. Státusz kód: 404

- a. „User not found” – az eredeti token sérült

Siker esetén:

1. Státusz kód: 200

- a. message: „Token updated”

- b. jwt: string – dekódolás után egy „interface”-t elégít ki (ld. ábra 5.1:7)

- II. Admin műveletekhez kapcsolódó végpontok (AdminController kezeli ezeket a végpontokat):

Minden API végpontra igazak a következők:

Header:

1. Authorization (autentikáció)

Típusa „Bearer” és tartalmaznia kell a JWT token

- a. Hibaüzenet:

- Státusz kód: 401 – „Unauthorized” (token hibás)

Hibaüzenetek:

1. Státusz kód: 403

- a. „You are not an admin!” – Vagy vendég vagy nem admin felhasználó

1. GET /admin /admins – adminok

Siker esetén:

1. Státusz kód: 200

- a. admins: User tömb (ld. ábra 5.1:8)

2. GET /admin/users – felhasználók listája

Siker esetén:

1. Státusz kód: 200

- a. admins: User tömb (ld. ábra 5.1:8)

3. GET /admin/reports – jelentések lista

Siker esetén:

1. Státusz kód: 200

- a. reports: Report tömb (ld. ábra 5.1:9)

4. DELETE /admin/reports/:id – jelentés törlése

Paraméter:

¹ Header tartalmaz olyan extra adatokat, mint például a válasz típusa, vagy autentikációs információk.

1. id – jelentés id-ja
Hibaüzenetek:
 1. Státusz kód: 404
 - a. „Report not found”
 Válasz (200):
 1. Message: „Report deleted”
5. PUT /admin/users/:id/ban – felhasználó tiltása
Paraméterek:
 1. id – felhasználó id
 Hibaüzenetek:
 1. Státusz kód: 404
 - a. „User not found.”
 Válasz (200):
 1. message: „User banned.”
 Socket event:
 1. banned – Ha elérhető (online) a felhasználó akkor kap erről kap értesítést, és amennyiben egy játékban volt, akkor a játék véget ér.
6. PUT /admin/users/:id/unban – felhasználó tiltásának
Paraméterek:
 1. id – felhasználó id
 Hibaüzenetek:
 1. Státusz kód: 404
 - a. „User not found.”
 Válasz (200):
 1. message: „User unbanned.”
7. PUT /admin/users/:userID/make-admin – felhasználó adminná tétele
Paraméterek:
 1. userID – felhasználó id
 Hibaüzenetek:
 1. Státusz kód: 403
 - a. „You cannot change super admin.” – szuper admin jogosultságait nem lehet változtatni
 2. Státusz kód: 404
 - a. „User not found.”
 Válasz (200):
 1. message: „User is now admin”
8. PUT /admin/users/:userID/remove-admin – felhasználó admin jogainak elvétele
Paraméterek:
 1. userID – felhasználó id
 Hibaüzenetek:
 1. Státusz kód: 403
 - a. „You cannot change super admin.” – szuper admin jogosultságait nem lehet változtatni
 2. Státusz kód: 404
 - a. „User not found.”
 Válasz (200):
 1. message: „User is no longer admin.”

9. PUT /admin/users/:id/add-remove-chat-restriction – felhasználó chat hozzáféréseinek módosítása
 - Paraméterek:
 1. id – felhasználó id
 - Hibaüzenetek:
 1. Státusz kód: 404
 - a. „User not found.”
 - Válasz (200):
 1. message: „User's chat restriction removed” vagy „User is now chat restricted.”
- III. Egyéni játékhöz kapcsolódó végpontok (CustomGameController kezeli ezeket a végpontokat):

Minden API végpontra igaz a következő:

Header:

 1. Authorization (autentikáció)

Típusa „Bearer” és tartalmaznia kell a JWT token

 - a. Hibaüzenet:
 - Státusz kód: 401 – „Unauthorized” (token hibás)
 1. GET /custom-games – egyéni játékok listája

Válasz (200):

 1. customGames: CustomGame tömb (ld. ábra 5.1:10)
 2. GET /wait-rooms/:roomUUID – várószoba adatai

Paraméterek:

 1. roomUUID – várószoba egyedi ID-ja

Hibaüzenetek:

 1. Státusz kód: 404
 - a. „Room not found”

Válasz (200):

 1. waitRoom: WaitRoom (ld. ábra 5.1:11)
 3. POST /wait-rooms/:roomUUID/:userID/kick – felhasználó kidobása a várószobából

Paraméterek:

 1. roomUUID – várószoba egyedi ID-ja
 2. userID – felhasználó ID

Hibaüzenetek:

 1. Státusz 403:
 - a. „Guest cannot make this request!”
 - b. „You are not the owner of this room” – nem jogosult arra a felhasználó, hogy kirakjon valakit
 2. Státusz 404:
 - a. „Room not found”
 - b. „User not found”

Válasz (200):

 1. message: „Player kicked”
- IV. Pályákkal kapcsolatos végpontok (FieldsController kezeli ezeket a végpontokat):

Minden API végpontra igaz a következő:

Header:

1. Authorization (autentikáció)
 - Típusa „Bearer” és tartalmaznia kell a JWT token
 - a. Hibaüzenet:
 - Státusz kód: 401 – „Unauthorized” (token hibás)
1. GET /fields – felhasználóhoz tartozó pályák listája
 - Hibaüzenetek:
 1. Státusz 403:
 - a. „Guest cannot make this request!”
 - Válasz (200):
 1. fields: FieldListResponseInterface tömb (ld. ábra 5.1:12)
2. GET /fields/:id – pálya részletes adatai
 - Paraméterek:
 1. id – pálya ID-ja
 - Hiba üzenetek:
 1. Státusz 403:
 - a. „Guest cannot make this request!”
 - b. „Acces denied” – olyan felhasználó próbálja lekérni a pálya adatait, akinek nincs ehhez jogosultsága
 2. Státusz 404:
 - a. „Field not found”
 - b. „User not found” – ha token formátumával nem volt baj, de felhasználó mégse létezik az adatbázisban¹
 - Válasz (200):
 1. field: FieldResponseInterface tömb (ld. ábra 5.1:12)
3. GET /fields/:id/show – megkülönböztetem azt az esetet mikor szerkesztésre van lekérve a pálya adatai, és mikor ténylegesen csak megjelenítésre
 - Paraméterek:
 1. id – pálya ID-ja
 - Hibaüzenetek:
 1. Státusz 404:
 - a. „Field not found”
 - Válasz (200):
 1. field: FieldResponseInterface tömb (ld. ábra 5.1:14)
4. POST /fields – új pálya létrehozása
 - Body:

```

1 interface FieldBodyInterface {
2     name: string;
3     field: {
4         dimension: { width: number; height: number };
5         players: PlayerInterface[];
6         objects: ObjectInterface[];
7     };
8 }

```

¹ Alapvetően ilyen nem fordulhat elő, csak abban az esetben, ha valaki tudja token kódolásához használt kulcsot és létrehoz magának egy token.

(PlayerInterface és ObjectInterface ld. ábra 5.1:14)

Paraméterek:

1. id – pálya ID-ja

Hibaüzenetek:

1. Státusz 403:
 - a. „Guest cannot make this request!”
2. Státusz 404:
 - a. „User not found” – ha token formátumával nem volt baj, de felhasználó mégsem létezik az adatbázisban
3. Státusz 400:
 - a. message: „Invalid Field”, details: string
 - Nincs extra valdiáció, csak annyi, hogy a FieldBodyInterface-t kielégítse és legyen minden a megfelelő típusú
 - Details – egy string melyben benne van minden hiba, pl:
 - Validation Error: "field.dimension" is required.
 - "field.players" is required. "field.objects" is required –
 - b. „Invalid Field” – a field nem elégíti ki a

Válasz (201):

1. field: FieldResponseInterface tömb (ld. ábra 5.1:12)

5. PUT /fields – pálya szerkesztése

Body:

```
1 interface FieldBodyInterface {
2     name: string;
3     field: {
4         dimension: { width: number; height: number };
5         players: PlayerInterface[];
6         objects: ObjectInterface[];
7     };
8 }
```

(PlayerInterface és ObjectInterface ld. ábra 5.1:14)

Paraméterek:

1. id – pálya ID-ja

Hibaüzenetek:

1. Státusz 403:
 - a. „Guest cannot make this request!”
2. Státusz 404:
 - a. „User not found” – ha token formátumával nem volt baj, de felhasználó mégsem létezik az adatbázisban¹
3. Státusz 400:
 - a. message: „Invalid Field”, details: string
 - Nincs extra valdiáció, csak annyi, hogy a FieldBodyInterface-t kielégítse és legyen minden a megfelelő típusú

¹ Alapvetően ilyen nem fordulhat elő, csak abban az esetben, ha valaki tudja token kódolásához használt kulcsot és létrehoz magának egy tokenet.

- Details – egy string, melyben benne van minden hiba, pl:
– Validation Error: "field.dimension" is required.
"field.players" is required. "field.objects" is required –
- b. „Invalid Field” – a field nem elégíti ki a

Válasz (201):

1. field: FieldResponseInterface tömb (ld. ábra 5.1:12)
6. DELETE /fields/:id – törli a pályát, „soft” és „hard delete”-et alkalmazva, tehát ha deletedAt null, akkor a törlés idejét megkapja, és legközelebbi törléssel végleg törölve lesz

Paraméterek:

1. id – pálya ID-ja

Hibaüzenetek:

1. Státusz 403:
 - a. „Guest cannot make this request!”
 - b. „Access denied” – olyan felhasználó próbálja lekérni a pálya adatait, akinek nincs ehhez jogosultsága
2. Státusz 404:
 - a. „Field not found”
 - b. „User not found” – ha token formátumával nem volt baj, de felhasználó még se létezik az adatbázisban¹

Válasz (200):

1. message: „Field deleted.” ábra 5.1:12
7. PUT /fields/:id/restore – visszaállít egy „soft deleted” pályát

Paraméterek:

1. id – pálya ID-ja

Hibaüzenetek:

1. Státusz 403:
 - a. „Guest cannot make this request!”
 - b. „Access denied” – olyan felhasználó próbálja lekérni a pálya adatait, akinek nincs ehhez jogosultsága
2. Státusz 404:
 - a. „Field not found”
 - b. „User not found” – ha token formátumával nem volt baj, de felhasználó mégsem létezik az adatbázisban

Válasz (200):

1. message: „Field restored.”

- V. Játékkal kapcsolatos végpontok (GameController kezeli ezeket a végpontokat):
Minden API végpontra igaz a következő:

Header:

1. Authorization (autentikáció)
Típusa „Bearer” és tartalmaznia kell a JWT token
a. Hibaüzenet:

- Státusz kód: 401 – „Unauthorized” (token hibás)

1. POST /games/:gameUUID/function – a játékos által megadott függvényt kell erre végpontra küldeni

Paraméterek:

1. gameUUID – játék egyedi azonosítója

Body:

1. fn – felhasználó által megadott függvény

Hibaüzenetek

1. Státusz 404
 - a. „Game not found”
2. Státusz 403
 - a. „It is not your turn” – Nem attól érkezett függvény, aki éppen soron következett.
3. Státusz 400
 - a. „You must submit a function” – függvény nem lett megadva
 - b. „Function already submitted”
 - c. „Invalid function”

Válasz (200):

1. message: „Function submitted.”

Válasz (200):

1. Message: „Game over” – erről socket esemény is érkezik, így azt elég ott kezelni.

Socket események:

1. „game over”, érkező adatok:
 - a. points: { leftSide:Point[], rightSide:Point[] }
 - b. message: „Game over, Winner is ...”
2. „receive function”
 - a. points: { leftSide:Point[], rightSide:Point[] }
 - b. damages: DamagesInterface (ld. ábra 5.1:13) – Az akadályokban okozott új sérülések találhatók itt. Függvény kiindulási pontjából nézve bal és jobb oldalon is lehet egy új sérülés.
 - c. length: number – a függvény hossza (pixeleken számolva, nem egységeken)

2. GET /games/:gameUUID – játék adatainak lekérése

Paraméterek:

1. gameUUID – játék egyedi azonosítója

Hibaüzenetek:

1. Státusz 404
 - a. „Game not found”
2. Státusz 403
 - a. „You are not in this game.”

Válasz (200)

1. Egy GameInterface-t kielégítő JSON objektumot küld

VI. Chat szóbával kapcsolatos végpontok (GroupChatController kezeli ezeket a végpontokat):

Minden API végpontra igaz a következő:

Header:

1. Authorization (autentikáció)
Típusa „Bearer” és tartalmaznia kell a JWT tokenet
 - a. Hibaüzenet:

- Státusz kód: 401 – „Unauthorized” (token hibás)

1. POST /group-chats/:roomUUID/mute/:userID – felhasználó némítása a chat szobában

Paraméterek:

1. roomUUID – chat szoba egyedi azonosítója
2. userID – felhasználó id-ja

Hibaüzenetek:

1. „Group chat not found.”
2. „You are not in this group chat.”

Válasz (200):

1. message: „User muted.”

2. POST /group-chats/:roomUUID/unmute/:userID – felhasználó némításának feloldása a chat szobában

Paraméterek:

1. roomUUID – chat szoba egyedi azonosítója
2. userID – felhasználó id-ja

Hibaüzenetek:

1. „Group chat not found.”
2. „You are not in this group chat.”

Válasz (200):

1. message: „User unmuted.”

3. GET /group-chats/:roomUUID/messages – chat szoba üzeneteinek lekérése

Paraméterek:

1. roomUUID – chat szoba egyedi azonosítója

Hibaüzenetek:

1. „Group chat not found.”
2. „You are not in this group chat.”

Válasz (200):

1. messages: MessageInterface tömb (ld. ábra 5.1:16)

4. GET /group-chats/:roomUUID/users-status – többi felhasználó státuszának lekérése

Paraméterek:

1. roomUUID – chat szoba egyedi azonosítója

Hibaüzenetek:

1. „Group chat not found.”
2. „You are not in this group chat.”

Válasz (200):

1. users:UserStatusInterface tömb (ld. ábra 5.1:17)

- VII. Barátokkal és más felhasználókkal kapcsolatos végpontok (FriendsController és UsersController kezeli ezeket a végpontokat)
Minden API végpontra igazak a következők:
- Header:
1. Authorization (autentikáció)
Típusa „Bearer” és tartalmaznia kell a JWT token
a. Hibaüzenet:
 - Státusz kód: 401 – „Unauthorized” (token hibás)
- Hibaüzenet:
1. Státusz 403
 - a. „Guest cannot make this request”
1. GET /friends – felhasználó barátai
Válasz (200):
 1. friends: User tömb (ld. ábra 5.1:8)
 2. GET /friends/online – felhasználó online barátai
Válasz (200):
 1. friends: User tömb (ld. ábra 5.1:8)
 3. GET /friends/requests – barát kérelmek
Hibaüzenet (404):
 1. „User not found.”
 Válasz (200)
 1. requests: FriendRequest tömb (ld. ábra 5.1:18)
 4. GET /friends/:id/chat – baráthoz kapcsolódó chat üzenetek lekérése
Paraméterek:
 1. Id – barát id-ja
 Hibaüzenet (404):
 1. „User not found.”
 2. „Friend not found.” – Ez azt jelenti, hogy a felhasználó nem létezik.
 3. „Friendship not found” – Nem áll fent barátság (itt előfordulhat, hogy a barátság megszűnt amíg a kérés fel lett dolgozva)
 Válasz (200)
 1. chat:
 - a. friendship_id: barátság kapcsolat azonosítója
 - b. messages: MessageInterface tömb (ld. ábra 5.1:16)
 5. PUT /friends/requests/:id/accept – barát kérés elfogadása
Paraméterek:
 1. Id – barátság kérelem id-ja
 Hibaüzenetek:
 1. Státusz 404
 - a. „User not found”
 - b. „Friend request not found.”
 2. Státusz 403
 - a. „You are not the receipient of this friendship request.”
 Válasz (200):
 1. message: „Friend request rejected.”
 6. DELETE /friends/requests/:id/reject – barát kérés elutasítása
Paraméterek:

1. id – barátság kérelem id-ja
- Hibaüzenetek:
1. Státusz 404
 - a. „User not found”
 - b. „Friend request not found.”
 2. Státusz 403
 - a. „You are not the receipient of this friendship request.”
- Válasz (200):
1. message: „Friend request accepted.”
7. DELETE /friends/:id – barát törlése
- Paraméterek:
1. Id – barátság kérelem id-ja
- Hibaüzenetek:
1. Státusz 404
 - a. „User not found”
 - b. „Friend request not found.”
 2. Státusz 403
 - a. „You can't delete a pending friendship.”
- Válasz (200):
1. message: „Friend deleted”
8. POST /friends/:id – barát kérelem küldése
- Paraméterek:
1. id – másik felhasználó id-ja
- Hibaüzenetek:
1. Státusz 404
 - a. „User not found”
 - b. „Other user not found”
 2. Státusz 403
 - a. „You can't add yourself as friend.”
- Válasz (200):
1. message: „Friend request sent.”
9. POST /users/:id/report – felhasználó jelentése
- Paraméterek:
1. id – másik felhasználó id-ja
- Body
1. description
- Hibaüzenetek:
1. Státusz 404
 - a. „User not found”
 2. Státusz 403
 - a. „You can't report yourself.”
 3. Státusz 400
 - a. „Description is required”
- Válasz (200):
1. message: „Report sent.”
10. POST /users/:id/block – másik felhasználó tiltása
- Paraméterek:

1. id – másik felhasználó id-ja

Hibaüzenetek:

1. Státusz 404
 - a. „User not found”
 - b. „Other user not found”
2. Státusz 403
 - a. „You can’t block yourself.”

Válasz (200):

message: „User blocked.”

11. DELETE /users/:id/unblock – másik felhasználó tiltásának feloldása

Paraméterek:

1. id – másik felhasználó id-ja

Hibaüzenetek:

1. Státusz 404
 - a. „User not found”
 - b. „Other user not found”
2. Státusz 403
 - a. „You can’t block yourself.”

Válasz (200):

1. message: „User unblocked.”

```

1 interface GuestUser {
2     readonly type: 'guest';
3     name: string;
4     id: string;
5     JWT_created_at: Date;
6     iat: number;
7 }

```

ábra 5.1:6

```

1 interface DecodedUser {
2     type: 'registeredUser';
3     id: number;
4     name: string;
5     email: string;
6     banned: boolean;
7     banned_reason: string;
8     is_admin: boolean;
9     role: string;
10    JWT_created_at: Date;
11    chat_restriction: boolean;
12    iat: number;
13 }

```

ábra 5.1:7

```

1 export interface User {
2     id: number;
3     name: string;
4     email: string;
5     banned: boolean;
6     banned_reason: string;
7     is_admin: boolean;
8     role: string;
9 }

```

ábra 5.1:8

```

1 interface ReportedBy{
2     id: number;
3     name: string;
4     email: string;
5 }
6 interface ReportedUser{
7     id: number;
8     name: string;
9     email: string;
10    chat_restriction: boolean;
11    banned: boolean;
12    banned_reason: string;
13 }
14 interface Report{
15     id: number;
16     description: string;
17     handled: boolean;
18     deletedAt: Date;
19     createdAt: Date;
20     reportedBy: ReportedBy;
21     reportedUser: ReportedUser;
22 }

```

ábra 5.1:9

```

1 interface CustomGame {
2     roomUUID: string;
3     userCount: number;
4     capacity: number;
5     fieldID: number;
6 }

```

ábra 5.1:10

```

1 type DecodedToken = DecodedUser
  | GuestUser;
2 interface WaitingRoom {
3     roomUUID: string;
4     chatUUID: string;
5     owner: DecodedToken;
6     players: DecodedToken[];
7     fieldID: number;
8     userCount: number;
9     capacity: number;
10 }

```

ábra 5.1:11

```

1 interface FieldListResponseInterface {
2   id: number;
3   name: string;
4   deletedAt: Date;
5   updatedAt: Date;
6 }

```

ábra 5.1:12

```

1 interface DamagesInterface {
2   leftSide?: null | {
3     location: PointInterface;
4     radius: number
5   };
6   rightSide?: null | {
7     location: PointInterface;
8     radius: number
9   };
10 }

```

ábra 5.1:13

```

1 interface PlayerInterface {
2   id?: number;
3   name?: string;
4   location: { x: number; y: number };
5   dimension: { width: number; height: number };
6   avoidArea: { location: { x: number; y: number }; radius: number };
7 }
8
9 interface ObjectInterface {
10   type: string;
11   location: { x: number; y: number };
12   dimension: { width: number; height: number };
13   avoidArea: { location: { x: number; y: number }; radius: number };
14 }
15 interface FieldResponseInterface {
16   id: number;
17   name: string;
18   field: {
19     dimension: { width: number; height: number };
20     players: PlayerInterface[];
21     objects: ObjectInterface[];
22   };
23 }

```

ábra 5.1:14

```

1  interface Dimension {
2      width: number;
3      height: number;
4  }
5
6  interface PlayerInterface {
7      id: number | string;
8      name: string;
9      location: PointInterface;
10     dimension: Dimension;
11     avoidArea: {
12         location: PointInterface;
13         radius: number
14     };
15 }
16 interface PlayerFieldInterface {
17     location: PointInterface;
18     dimension: Dimension;
19     avoidArea: {
20         location: PointInterface;
21         radius: number
22     };
23 }
24 interface ObjectInterface {
25     type: string;
26     location: PointInterface;
27     dimension: Dimension;
28     avoidArea: {
29         location: PointInterface;
30         radius: number
31     };
32     damages: {
33         location: PointInterface;
34         radius: number
35     }[];
36 }
37 interface FieldInterface {
38     id: number;
39     name: string;
40     field: {
41         dimension: Dimension;
42         players: PlayerFieldInterface[];
43         objects: ObjectInterface[];
44     };
45 }
46
47 interface GameInterface {
48     players: PlayerInterface[];
49     objects: ObjectInterface[];
50     currentPlayer: PlayerInterface;
51     lastFunction: string | undefined;
52     field: FieldInterface;
53     uuid: string;
54 }

```

ábra 5.1:15

PointInterface egy x-et és y-t tartalmaz,
mindkettő egész szám

```

1  interface UserInterface {
2      id: number | string;
3      name: string;
4  }
5  interface MessageInterface {
6      from: UserInterface;
7      message: string;
8  }

```

ábra 5.1:16

```

1  interface UserStatusInterface{
2      id: number;
3      name: string;
4      blocked: boolean;
5      muted: boolean;
6      isFriend: boolean;
7  }

```

ábra 5.1:17

```

1  interface FriendRequest {
2      id: number;
3      from: {
4          id: number;
5          name: string;
6      };
7  }

```

ábra 5.1:18

Websocketek

Socketek kezelésére a SocketIO használok, ami, mind backend mind frontend oldalra biztosít csomagot. A socket szerveret az app.js fájlban inicializálom, ahol hozzáadok middleware¹, ami azt nézi, hogy JWT token megtalálható-e a headerben, illetve, hogy helyes-e. Majd a dekódolt JWT tokenet hozzáadja a sockethez (socket.decoded).

Egy eseményt úgy kell indítani, hogy socketre meghívjuk az „emit” függvényt, ez paraméterül várja az esemény nevét, és opcionálisan egyéb paramétereket, amiket JSON objektumba kell rakni. Eseményre az „on” metódussal lehet hallgatni, az első paramétere az esemény neve, a második paraméter pedig egy függvény, ami kezeli majd ezt az eseményt, ennek a függvénynek a paramétere minden esetben az a JSON objektum, ami az eseményhez tartozik. Socketekhez tartozik egy egyedi id, és ezzel meg lehet őket különböztetni backend oldalon őket, továbbá lehet eseményt az ID-val egy socketnek (socket.to(socketID)). A „join” függvénnyel létre lehet hozni szobákat, ehhez a socketre meg kell hívni és egy szoba nevet/ID-t kell megadni, és szobába is lehet eseményeket küldeni, socket.to(„szoba név”).

Backend oldalon figyelt események (frontend oldalon indított események)

1. connection

SocketConnectionService kezeli, ez az esemény mindig akkor van, amikor egy új socket csatlakozott. SocketConnectionService szerepe az, hogyha egy játékban volt egy felhasználó, de onnan kilépett, mert esetleg elment az internet kapcsolata, vagy bezárta a lapot véletlen, akkor 10 másodperce van vissza csatlakozni, SocketConnectionService.userConencted függvény egész egyszerűen hozzáadja RuntimeMaps.onlineUsers-hez a felhasználót.

```
1 RuntimeMaps.onlineUsers.set(socket.decoded.id, { user: socket.decoded,
  socketID: socket.id, currentURL: '' });
```

2. route change

Mindig akkor kell küldeni, ha felhasználó másik oldalra navigált, és itt szükség van a JSON objektumban egy url adatra, hogy melyik oldalra navigált – frontend oldalon ugyanazokat a végpontokat kell tartani, amik az apihoz vannak –. Url lehet például: „”, '/fields’ stb. Ezért az eseményért a SocketConnectionService.userNavigated metódus felel, mely megnézi, hogy a felhasználó játékból lépett-e ki, ha igen véget vet a játéknak. Ha pedig várószobából lépett ki, akkor onnan kilépteti, illetve, ha várószoba tulajdonosa lépett ki, akkor törli a várószobát.

3. create custom game

Mikor a felhasználók új egyéni játékot szeretnének létrehozni, ezt az eseményt kell küldeni, az esemény JSON objektumában küldeni kell a kiválasztott pálya id-t (fieldID), hogy privát-e az indítani kívánt játék (isPrivate) és a meghívni kívánt barátok listája (friendIDs). Eseményt a CustomGameController.createCustomGame függvény kezeli, ellenőrzi, hogy

¹ Még mielőtt a kliens csatlakozna szerverhez a middleware lefut, és ha nem talált hibát, akkor engedni csatlakozni.

lett-e megadva pálya ID, ha nem akkor „error” eseményt küld. Majd létrehoz egy várószobát („WaitingRoom”), és „waiting room created” eseményt küld. Meghívott barátoknak pedig kiküldi a meghívókat.

4. join custom game

Egyéni játékhoz¹ csatlakozáshoz ezt az eseményt kell indítani, az eseményhez szükséges küldeni a várószoba ID-ját („roomUUID”). Az eseményért a CustomGameController.joinWaitingRoom függvény felel. Ha nem találta meg a várószobát, vagy ha megtelt a várószoba, akkor egy „error” eseményt küld. Ha minden jól ment, akkor a szobának a „user joined waiting room” event kerül kiküldésre és a felhasználónak pedig a „waiting room joined” esemény.

5. start custom game

Az egyéni játék tulajának kell ezt az eseményt indítani, az esemény nem vár semmilyen paramétert. CustomGameController.startGame metódus kezeli ezt az eseményt, lényegében csak a GameController.createGame függvényét hívja meg, ennek szüksége van socketekre (sockets), amik benne vannak a várószobában és pálya id-ra (fieldID). Létrehoz egy játékot az adott pályával, és minden felhasználót belerak, a felhasználókhoz tartozó socketek pedig „joined game” eseményt kapnak.

6. reject invite

Ha a felhasználó elutasít egy meghívást ezt az eseményt kell küldeni, paraméterül a meghívást kell küldeni (invite), ami tartalmaz egy inviterID-t és egy invitedID-t. Itt csak annyi történik, hogy a RuntimeMaps.invites halmazból (Set) törlésre kerül ez a meghívás (invite).

7. send chat message

Mikor a felhasználó egyik barátjának küld üzenetet, akkor ezt az eseményt kell küldeni, paraméterül egy message és egy friend_id adat kell. Az eseményért a FriendChatController.sendMessage felel. Ha nem találta meg a barátot, vagy ha vendég akar ilyen üzenetet küldeni – ilyen alap esetben nem történhet, de le van kezelve –, akkor „error” esemény kerül kiküldésre. Ha minden jól ment, de még nem lett chat létrehozva ehhez a barátsághoz, akkor létrehozza, különben pedig csak hozzá adja az új üzenetet. Ha barát fent van akkor egy „receive message” eseményt küld a baráthoz tartozó socketre.

8. set seen

Ezt az eseményt akkor kell küldeni, ha a felhasználó megnyitja a chatet vagy, ha nyitva van és új üzenetet kap, akkor minden esetben. Az eseményhez a barát azonosítóját (friend_id) kell küldeni, és a FriendChatController.setSeen függvény kezeli az eseményt. Ebben az esetben is, ha nem találta meg a barátot, vagy ha vendég akar ilyen üzenetet küldeni, akkor „error” esemény kerül kiküldésre. Abban az esetben, ha minden jól ment, akkor a minden üzenetnek a „seen” értéke „true” lesz – természetesen csak azoknál az üzeneteknél, amiket a felhasználó kapott.

9. join wait list

Az eseményhez nem kell semmilyen paramétert küldeni. WaitListController.joinWaitList függvény kezeli ezt az eseményt, ha vannak

¹ Az egyéni játék és várószoba gyakorlatilag szinonimák, mert ha egy egyéni játék el lett indítva, akkor onnantól rendes játékként van kezelve.

elegen a várólistán (legalább 2-en), akkor a meghívja a createGame függvényt a GameControllerből, és átadja a várólistában szereplő első 2-4 felhasználót, és egy random pályával létrehoz egy játékot.

10. leave wait list

Nem igényel semmilyen paramétert ez az esemény, és a WaitListController.leaveWaitList kezeli, ami csak annyit csinál, hogy leveszi a várólistáról felhasználót.

11. send group chat message

Paraméterként az üzenetet (message) várja az esemény, majd kiküldésre kerül az üzenet minden olyan felhasználónak, akik szeretnének üzenetet kapni az adott felhasználótól, illetve csak abban az esetben küldi el az üzenetet, ha a felhasználó tud üzenetet küldeni.

GroupChatController.sendMessage függvény kezeli, és meghívja az adott chat szobához tartozó osztály sendMessage függvényét. Ez az, ami lényegében kiküldi az üzenetet a felhasználóknak.

12. disconnect

Ez az esemény a connection-höz hasonlóan automatikusan indul, amikor a felhasználó kilép valamilyen okból kilépett – ez lehet a kijelentkezés is. Ezért az eseményért a SocketConnectionService.userDisconnected metódus felel, a felhasználókat kilépteti a várószobából, várólistáról és játékból, de játékot és várószobát (ha tulaj lépett ki), csak 10 másodperces késleltetéssel ellenőrizi, hogy időközben visszacsatlakoztak-e, és ha nem, akkor törli ezeket. Így van ideje a felhasználónak visszalépni.

Backendén küldött események (frontenden kell figyelni ezeket)

1. banned

Ez az esemény akkor történik meg, ha egy admin tiltja a felhasználót, ilyenkor a felhasználót ki kell jelentkeztetni. Esemény paraméterben megy az üzenet (message: „You have been banned”)

2. error

Ez az üzenet több helyről is érkezhetsz, ezért frontenden érdemes a „root” komponensben megjeleníteni. Minden esetben egy „message” és egy „code” adat van küldve.

3. waiting room created

Ez egy vissza jelzés a kliensnek, hogy a létrehozott várószoba elkészült és lehet csatlakozni hozzá. Paraméterül megy egy „roomUUID”, ami várószoba azonosítója, egy groupChatUUID, ami pedig a hozzátartozó chat szoba azonosítója.

4. waiting room joined

Visszajelzés a kliensnek, hogy sikeres csatlakozott a várószobához.

5. user joined waiting room

Ha egy új felhasználó csatlakozott a várószobához, akkor erről értesíti a többi felhasználót. Ilyenkor minden adatot le kell kérni újra a várószobával kapcsolatban.

6. user left wait room

Ha egy felhasználó kilépett, akkor van ez az esemény küldve, és ugyanúgy le kell kérni a várószoba adatait.

7. wait room owner left

Akkor következik be, ha tulajdonos lépett ki, ilyenkor vissza kell navigálni a főoldalra.

8. receive invite

Egy barát meghívta a felhasználót egy egyéni játékba. Az eseménnyel a barát és az egyéni játék azonosítója érkezik ({ inviter:number, customGameUUID })

9. receive message

Amikor egy barát új üzenetet küld. Barát azonosítója („from”), a küldött üzenet és a („message”) és hogy látta-e („seen”), utóbbi természetesen „false”.

10. game ended

Abban az esetben történik, ha egy felhasználó kilépett a játékból, ehhez megy egy üzenet („message”): `\${user} left the game.`

11. game over

Akkor következik be, ha valaki megnyerte a játékot, ehhez is megy egy üzenet, amiben benne van, hogy ki nyert („message”)

12. receive function

A beküldött függvény pontjait küldi („points”), az új sérüléseket („damages”) és a függvény hosszát pixelekből számolva („length”). A pontok és a sérülések is egy JSON objektum, amiben egy „leftSide”, és egy „rightSide” mező van.

```
1 points: {  
2   leftSide: PointInterface[];  
3   rightSide: PointInterface[]  
4 };
```

```
1 damages: {  
2   leftSide?: { location: PointInterface; radius: number } | null;  
3   rightSide?: { location: PointInterface; radius: number } | null;  
4 };
```

13. joined game

Hogyha egy játék sikeresen létre lett hozva a játékosokkal, akkor arról a játékosok értesülnek. Játék azonosítóját küldi a „room” mezőbe, a játékban lévő játékosok adatait pedig a „players”-be, illetve a pálya adatait pedig a „field”-be.

14. new user joined grout chat

Ha új felhasználó csatlakozott a chat szobába erről érkezik egy értesítés. Adatok nem érkeznek ezzel az eseménnyel.

15. user banned

Ez az esemény is a játék végét jelenti, ezt azok a felhasználók kapják, akik épp játékban vannak egy olyan játékosal, aki ki lett tiltva.

16. receive group chat message

Mindig amikor valaki új üzenetet küld, ezt az üzenetet csak azok kapják meg, akik akarnak üzenetet kapni a küldő játékosától, tehát nincs némítva vagy tiltva.

17. owner left

Értesítés arról, hogy az egyéni játék tulajdonosa kilépett.

18. kicked from wait room

Akkor következik be, ha a felhasználó ki lett rakva a várószobából, ilyenkor vissza kell irányítani a felhasználót a főoldalra.

Segéd osztályok (utils)

1. Obstacle (ld. ábra 5.1:19)

Az Obstacle osztály a pályán lévő akadályokat reprezentálja, kétféle akadály lehet, téglalap és ellipszis. Minden akadály esetében máshogy van eldöntve, hogy az akadályt eltalálta-e a függvény, de ami közös, hogy egy f függvény esetén az $(x;f(x))$ koordinátát veszik paraméterül. Ellipszis esetén egyszerű a képlet, legyen az Ellipszis középpontja $p:(px;py)$ és függvény egy pontja $fp:(fx;fy)$, továbbá W az ellipszis szélessége és H az ellipszis magassága, ekkor fp benne van ellipszisben, ha:

$$\frac{(px-fx)^2}{W^2} + \frac{(py-fy)^2}{H^2} \leq 1$$

De még mielőtt ez az ellenőrzés megtörténne, meg kell nézni, hogy az adott pont benne van-e bármilyen sérülésben – más Obstacle példányok sérüléseit is ellenőrizni kell. Mivel a sérülések csak egy kör, így ott triviális, hogy ott hogyan történik az ellenőrzés.

Téglalapok esetében már bonyolultabb a képlet. Ebben az esetben szükséges egy Vector osztály definiálása, valamint minden csúcshoz a koordinátáját kikell számítani. A csúcsokon végig iterálok, és minden esetben az i . csúcsból az $i+1$. csúcsba és az i . csúcsból a vizsgálni kívánt pontba mutató vektort kell kiszámolni – utolsó csúcs esetén pedig az első csúcsba mutató vektort. Majd ezeknek vektoriális szorzatát kell venni, és abban az esetben, ha a vektoriális szorzat harmadik komponens értéke mindig nagyobb, mint 0, vagy mindig kisebb, mint 0, akkor a pont benne van a téglalapban.

2. FuncCalculator (ld. ábra 5.1:20)

Ez az osztály felel azért, hogy a függvény ellenőrizve legyen, hogy a beírt függvény a program számára értelmezhető állapotba kerüljön, és hogy kiszámolja mindkét irányban a függvény pontjait.

Függvény módosító metódusok:

1. –

Az alap esetre nincs külön függvény, ezt az `replaceAll` futtatja, és minden használható függvény (`sin`, `cos`, `sqrt`, `log`, `pow` és `abs`) elé beilleszti a „`Math.`” szövegrészletet.

2. `replacePowerOperator`

Minden „`^`” jelet lecseréli a „`**`” operátorra, ami JS-ben hatványozást jelenti, pl. $2^3 = \text{Math.pow}(2,3) = 2**3$.

3. `replaceEWithMathE`

Minden „`e`”-t lecserél `Math.E`-re.

4. `replaceAbsWithMathAbs`

Az abszolút értékeket konvertálja, ehhez egy reguláris kifejezést használok (`/\|([^\|]+)\|/g1`). Ez a JS regex megkeres minden olyan szövegrészt, ami abszolút értékben van, és legalább egy karaktert tartalmaz, ami nem „`|`”-jel. Ezzel a regex-el futtatom a JS gyári string függvényét, a `replaceAll`-t, és minden talált szövegrészt lecseréli a „`Math.abs($&)`”-re, „`$&`” a regex által megtalált szövegrészeket jelöli. Majd minden „`|`” jelet kicserél egy üres stringre.

5. `replaceSqrtOperator`

Egy gyökvonás jel és kezdő zárójeltől egy vég zárójelig megkeres minden szövegrészletet (`/\|([^\|]+)\|/g1`) és kicseréli

„Math.sqrt(\$&)”-re. Ezután pedig az összes $\sqrt{}$ jelet üres stringre cseréli.

6. modifyNegativeNumbers

Az összes szorzás vagy hatvány előtti vagy utáni „X”-et lecserél egy „(X)”-re. Továbbá minden „-x”-et egy „-(X)”-re.

7. insertMultiplicationOperator

Az X elé, ha egy szám vagy „)” előzi meg, beilleszt egy szorzás jelet, és hasonlóan tesz, ha X után van egy szám vagy „(” jel.

Egyéb metódusok az osztályban:

1. f(x : number) : number | never¹ – „never” visszatérési érték hibát jelent –

A függvény JS kód formátumú átalakításokra azért van szükség, mert a JS-ben van egy metódus, az „eval”, mellyel egy szöveggént megadott JS kifejezést le lehet futtatni, ha valami szintaktikai hiba van, akkor pedig hibát ad ki.

A FuncCalcutatornak szüksége van az aktuális játékos pozíciójára, mert a (0;0) koordinátát ehhez igazítja, a függvényben minden „X” karaktert cserél $\frac{x-zeroX}{ratio}$, ahol x egy aktuális pont a [0;1000] intervallumon – frontenden a canvas elem maximum szélessége 1000 –, zeroX pedig az aktuális játékos X koordinátája, ratio pedig arány, hogy egy egység hány pixel, ez 35, tehát a helyettesítési érték a [0;1000/35] intervallumban van. Majd a kiszámolt értéket vissza kell konvertálni – aránnyal való szorzás –, továbbá az Y tengelyen is a játékoshoz kell igazítani a kiszámolt értéket: $round(zeroY - f(x) * ratio) - f(x)$ ebben az esetben eval(x)-et jelent –, round egy kerekítést jelent.

2. async firstValidPoint() : Promise<PointInterface | null>

Előfordulhat, hogy egy függvény a x=0-ban nem értelmezett, de ennek ellenére nem sért meg semmilyen ellenőrzési hibát sem, ilyen pl. a log(x). Ezért ez a függvény megkeresi az első olyan X koordinátát, amiben értelmezett a függvény, és az aktuális játékos bázisában van. Ha nincs ilyen pont, akkor a megadott függvény nem elfogadott.

3. async calculateRightSidePoints(): Promise<PointInterface²[]>

Ha a firstValidpoint() nem null, akkor az x-től kezdve 1000-ig kiszámolja a pontokat. Előfordulhat, hogy egy pontban végtelen értéket számol, pl. a JS log(0)-ra -Infinity értéket számol, ilyenkor $+\infty$ esetén az y értéke 700 lesz – hiszen a canvas (0;0) koordinátája a bal felső sarokban van – és a $-\infty$ esetén pedig 0 lesz az y. Majd az így keletkezett egyenest a Line osztály segítségével több kisebb pontra kell osztani.

4. async calculateLeftSidePoints() Promise<PointInterface[]>

A „calculateRightSidePoints” függvényhez képest annyiban más, hogy a x...0 irányba számolja az értékeket.

5. isSatisfiesAnyInvalidCase(): boolean

¹ TS-ben a „never” típus hiba dobást jelen.

² Egy „x” és „y” értéket tartalmaz.

Az osztályban található egy `invalidCases` tömb, melynek minden eleme egy JSON objektum, egy regex, és egy `message` attribútummal. Rossz esetek lehetnek:

1. Tangens függvény, ehhez a regex, megnézi, hogy tartalmaz-e „tan(” szövegrészt.
2. X-szel való osztás esetén pedig megnézi, hogy a hányadosban szerepel-e bármilyen x-et tartalmazó kifejezés.
3. Szorzás jel nélküli szorzáshoz megnézi, hogy zárójel előtt vagy után van-e szám, illetve, hogy két zárójel egymás mellett van-e közvetlenül pl. „2(x+2”, „(x+2)2” vagy „(x+1)(x+2)”.

Ezekre az rossz esetekre ellenőrzi a függvényt.

6. `isValidFunction()`: boolean

Ellenőrzi, hogy függvény meg van-e adva, és hogy `fristValidPoint()` vissza térési értéke ne null legyen, illetve, hogy `isSatisfiesAnyInvalidCase` értéke false.

3. `Game` (ld. ábra 5.1:21)

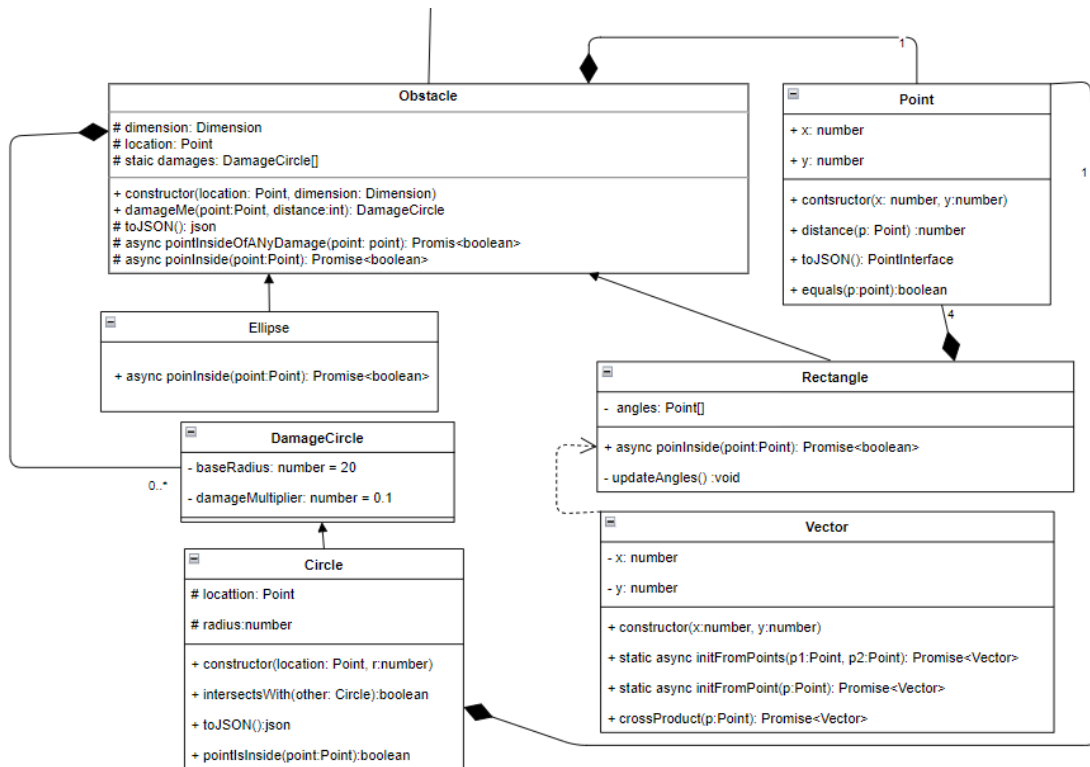
Függvények:

1. `changeCurrentPlayer`
2. `async submitFunction(func : string)`

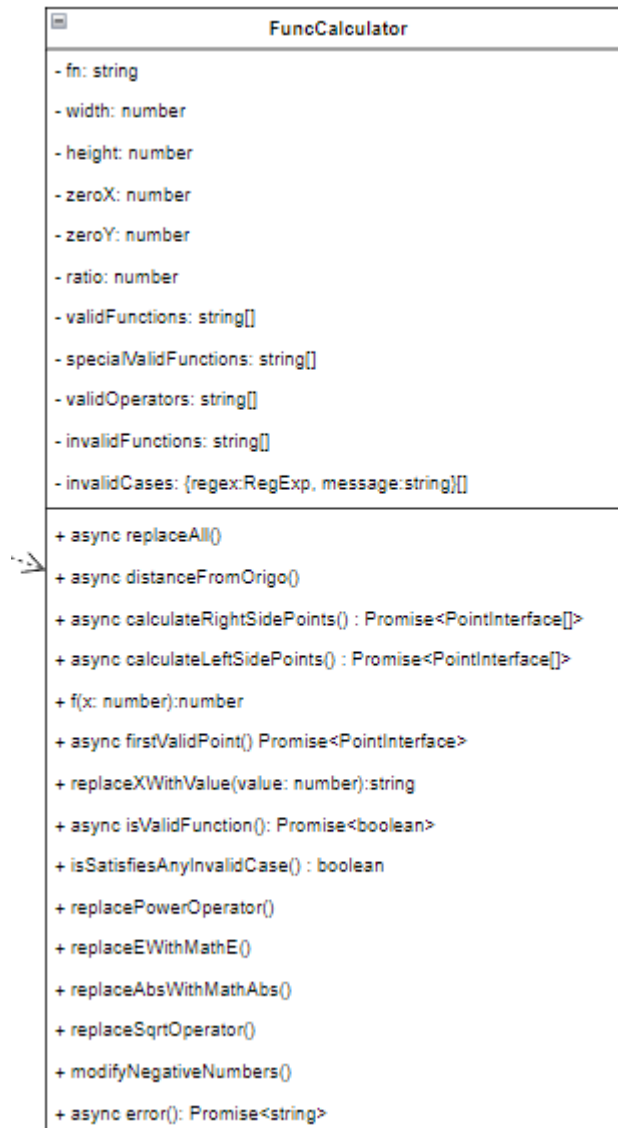
Ellenőrzi, hogy az aktuális játékos által lett-e már beküldve a függvény, ha igen hibát dob, majd megnézi a `FuncCalculator` osztállyal, hogy helyes legyen a függvény, abban az esetben, ha nem jó akkor hibát dob.

3. `async calculateFunctionPoints`

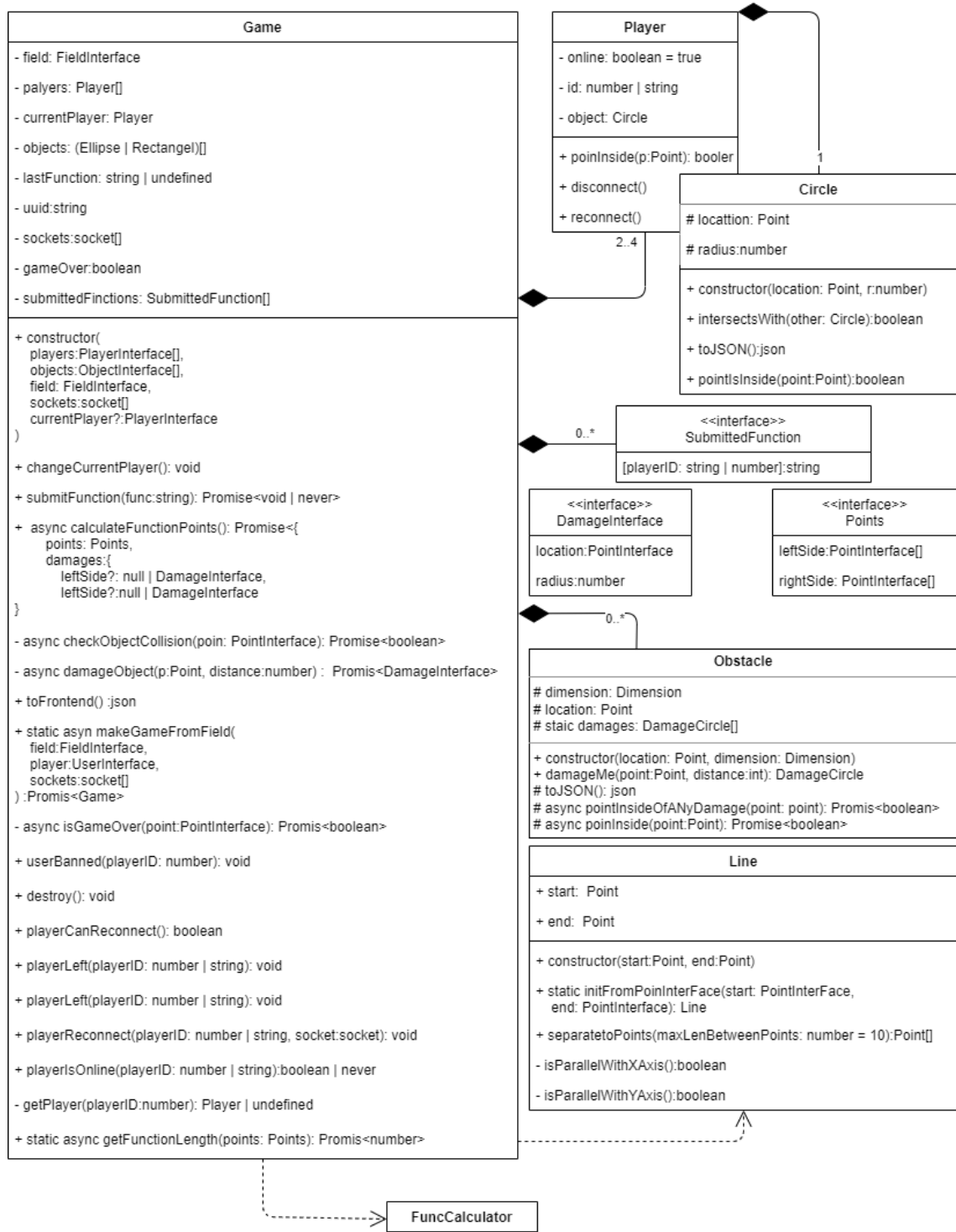
Meghatározza a függvény pontjait a `FuncCalculator` segítségével mindkét irányba. Miután meghatározta a pontokat végig megy minden összesen, és ellenőrzi, hogy az adott pont benne van-e egy akadályban, vagy játékos bázisában. Ha akadályba, vagy játékos bázisba ütközött a függvény, akkor az ütközési pont után következő pontokat levágja. A pontok mellett még az újonnan létrejött sérüléseket is meghatározza.



ábra 5.1:19



ábra 5.1:20



ábra 5.1:21

Használt külső csomagok

1. bcryptjs

Jelszavak titkosítására használom, ehhez a hashSync függvényt kell meghívni a jelszavakra.

2. chalk

Console.log()-t egészíti ki úgy, hogy lehessen színesen logolni a terminálra pl. Console.log(chalk.red(Error:))

3. jest

Tesztelő keretrendszer NodeJS projektekhez, a szintaxisa ugyanaz, mint a NodeJS beépített teszt környezete. Annyiban tér el, hogy a jest minden olyan fájlt meg keres a projektben, ami illik erre: *.test.js

4. joi

Komplexebb JSON objektumok ellenőrzésére használható, ehhez létre kell hozni egy sémát: Joi.object() vagy akár Joi.string(), ha nem objektum validálásra akarjuk használni. Majd létrehozott sémára a „validate” függvényt kell meghíni úgy, hogy annak paramétere a kapott adat (`schema.validate(field)`). Objektum esetén fel kell sorolni a lehetséges kulcsokat, majd minden kulcsra meg lehet határozni, hogy milyen típusnak kell lenni, és milyen szabályok érvényesek rá.

5. jsonwebtoken

A .env-ben meghatározott JWT_SECRET és a JWT_ALGO segítségével létrehoz egy JWT token. Ehhez a „sign” metódust kell hívni, melynek első paramétere a JSON objektum, amit kódolni kell, 2. paraméter pedig JWT_SECRET, ennek a segítségével ismeri fel a token, illetve a 3. paraméter különféle opciókat tartalmaz, itt csak az „algorithm” opció van megadva.

6. express-jwt

A kliens felől érkező JWT token validálja, megnézi, hogy a token azzala „titokkal” lett-e kódolva, ami a JWT_SECRET környezeti változóban van (.env). Ha nem, akkor 401 Unauthorized hibát dob. Minden API végpont esetén, ami igényli az autentikációt, ott az ezzel a csomaggal definiált middleware fut le.

7. nodejs-better-console

Felülírja a natív js console műveleteket, minden esetben kiírja színessel, hogy melyik fájl, hanyadik sorában történt a kiírás, majd az üzenetet egy új sorban írja ki. Ehhez a server.js-ben írtam csak felül console-t, hiszen így a futtatáskor az egész projektben a felül írt változat lesz elérhető (overrideConsole()-t kell csak meghívni).

8. node-mailer

Ennek a „package”-nek a segítségével lehet csatlakozni az SMTP szerverhez és leveleket küldeni. Szükség van egy mailOptions-re, amibe kell a „from” – küldő e-mail cím, „to” – címzett e-mail címe, „subject” – e-mail tárgya és „html” – az e-mail tartalma html formátumban. továbbá mailConfigra, „host”, „port”, „secure_connection”, „auth.user”, „auth.pass” és „tls”. TLS opciót leszámítva az összes konfiguráció információ a .env fájlból érkezik. Miután ezek megvannak, létre kell hozni egy transporter: „createTransporter(mailConfig)”, majd meg lehet hívni a transporter sendMail metódusát, melynek első paramétere a mailOptions és egy callback, melynek 2 paramétere van, „error” és „info”, ha végzett, akkor ez a callback fut le. Itt lehet kezelni a hibát és a sikeres küldést is.

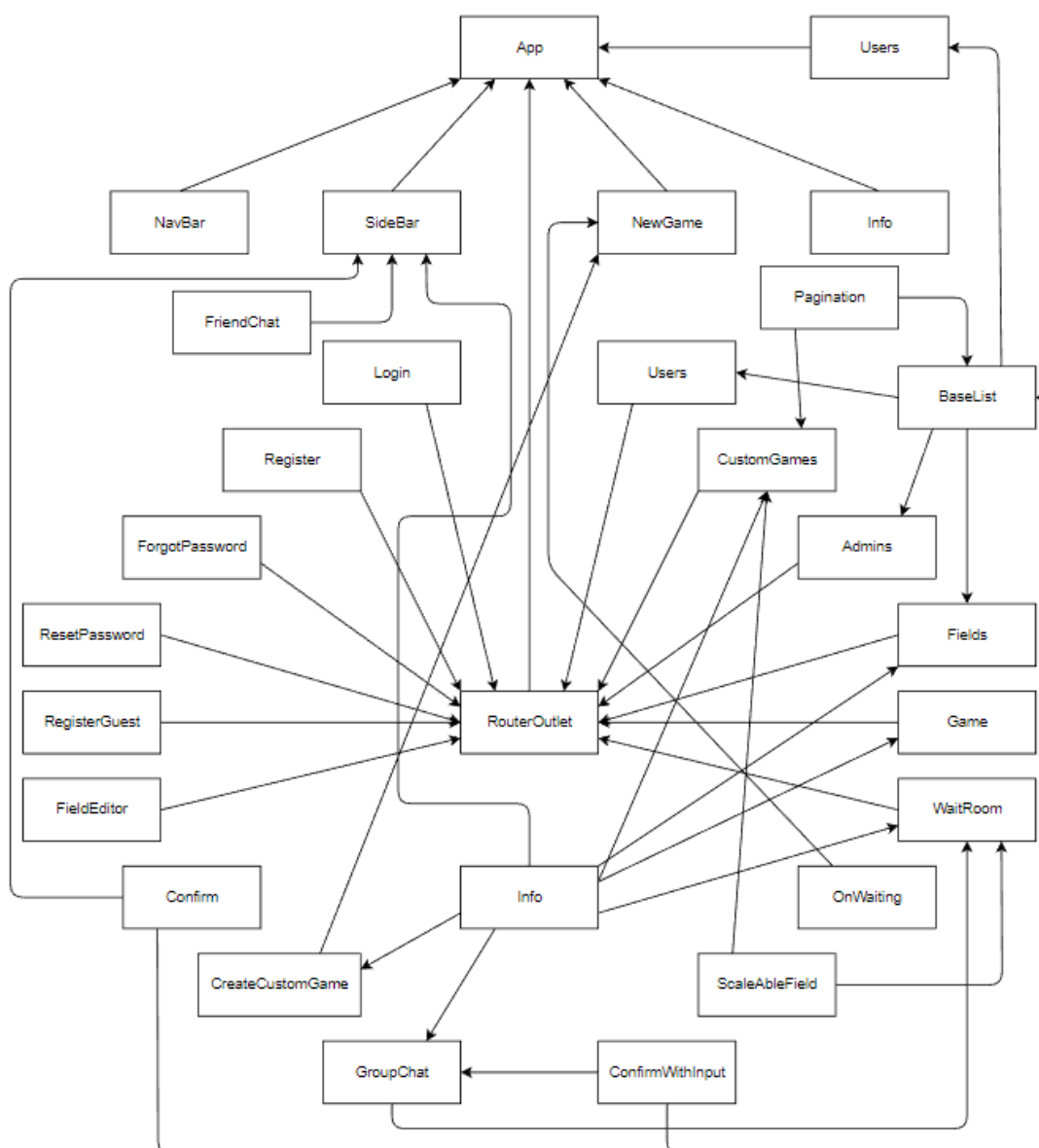
5.2 Frontend

Anugularban van egy root komponens, ez az App, az itt definiált komponensek mindig megjelennek ilyen pl. a navigációs sáv (NavBar). Az App komponensben van egy RouterOutlet komponens, ez mindig azt az aktuális komponenst jeleníti meg, ami az URL

szerint definiálva van. Ehhez úgy lehet komponenseket adni, hogy meg kell határozni az útvonal-komponens párokat az „app-routing.module.ts” fájlban.

Komponenseket lehet generálni CLI segítségével, „ng generate component <name>”, a komponens nevét „kebab case” használatával kell megadni, tehát például a RegisterGuest komponenst „register-guest”-nek kell elnevezni. Ez létrehoz egy mappát, benne 4 különböző fájlal, egy css-el (az én esetemben scss), egy html-el, egy TS fájlal és a komponenshez tartozó teszt fájlal. Minden komponenshez csak template (html) tartozhat, de több css/scss fájlt is hozzá lehet csatolni. Html elemeket és komponenseket el lehet rejteni, ehhez a hozzá kell adni a „tag”-en belülre az *ngIf attribútumot és meg kell adni egy feltételt, és csak akkor jelenik meg a komponens, ha feltétel igaz. Elemek ismétléséhez az *ngFor=”let item of items” attribútumot lehet használni. Minden olyan változó/függvény, ami publikus a ts-fájlban, az elérhető a html fájlban is, ehhez csak dupla kapcsos zárójelbe kell írni ({{ }}), kivétel az *ng kezdetű attribútumok, ezeknél automatikus. Angularban a változó értékek azonnal frissülnek, tehát nincs szükség arra, hogy a DOM manuálisan frissítve legyen.

Az oldalhoz definiáltam „service” osztályokat, ezekben olyan kódrészletek vannak, amiket több helyen is fel kellene használni. Ezek használata nagyon egyszerű, elég a komponens konstruktorának egy olyan paramétert adni, ami valamilyen service típusú, az angular az osztály példányosítását elintézi (pl. constructor(private jwt: JwtService)).



3. tooltip – segéd üzenet, ha a felhasználó fölé vitte az egeret,
4. confirmRequired – egy callback, paramétere az adott adat, és meg kell határozni, hogy a kiválasztott adathoz, mikor kérjen megerősítést,
5. confirmType – ez lehet DEFAULT vagy WITH_INPUT, DEFAULT esetén egy sima „yes or no” visszaigazolást kér, WITH_INPUT-nál egy bementi mezővel ellátott visszaigazolás jön elő
6. confirmData – leírás, ami megjelenik felugró ablakon
7. visibleWhen – callback aminek paramétere az adat, és az éppen belépett felhasználó

Ezek alapján a BaseListnek a következő adatokra van szüksége:

1. collectionSize – hány adat van
 2. pageSize – mennyi adat jelenjen meg egy oldalon
 3. page – melyik oldal jelenjen meg elsőnek
 4. headers – Header tömb
 5. data – az adatok, amiket meg kell jeleníteni, ennek típusa any
 6. singularActions – azok a műveletek, amiket csak egy adatra lehet végrehajtani egyszerre (Action)
 7. pluralActions – olyan műveletek, amiket több adatra végre lehet hajtani
- Amikor egy felhasználó rákattint egy műveletre, akkor a BaseList egy eseményt indít, erre a szülő komponens hallgat. BaseList elküldi a műveletet, az adatokat (data:any[]) – tömbben küldi abban az esetben is, ha csak egy adat lett kiválasztva – és a confirmInput, ami lehet undefined

5.3 Tesztelés

Tesztelési tervem elég egyszerű, backend tesztelek csak, hiszen frontenden csak megjelenítés történik, és ehhez a API „request”-eket küldök, és ezeknek a választ ellenőrzöm, igyekeztem minden esetet lefedni az API végpontok esetében, hogy hibás adatokra megfelelő hibát dob-e, illetve, ha minden jól ment, akkor az elvárt adatok érkeztek-e. Hasonlóan történik a socketek tesztelése is. Elindítok egy eseményt, és ellenőrzöm, hogy a többi kliens megkapta-e.

API kérések ellenőrzésére a jest és supertest csomag van segítségemre, „supertest”-el egyszerűen lehet elindítani az express servert, és amint lefutottak tesztesetek leállítja a szervert. Socketek esetében manuálisan kell indítani, és leállítani a szervert.

API teszteléshez be kell importálni a „supertest”-t mellé az „app”-t, amiben az express szervert definiáltam, és csak paraméterül kell adni a supertestnek az appot (supertest(app)), ez visszatér egy „request”-el, amire lehet küldeni az API kéréseket.

A tesztesetek az „npm run test” vagy „npm run win:test” paranccsal futtathatóak. Van lehetőség tesztelésről egy elemzést generálni, „npm run coverage” vagy „npm run win:coverage”, ami megtekinthető a coverage/Icov-report mappában az index.html megnyitásával, vagy <https://torlev.web.elte.hu/szakdolgozat> oldalon megtekinthető.

6 Összefoglalás

Szakdolgozatom összességében, tehát egy játékot valósít meg, amiben matematikai függvények képével kell eltalálni más játékosokat. A függvényt különféle módokon ellenőrizni, program által értelmezhető formátumra hozni, majd kiszámolni az értékeit és továbbítani a felhasználóknak. Játék mellett moderálási rendszer kiépítése, és felhasználók közötti különböző kapcsolatok megvalósítása. Illetve különféle felhasználói típusok kezelése.

A függvényellenőrzése két lépcsőből áll, 1. reguláris kifejezések segítségével ellenőrzöm, hogy lett-e olyan függvény megadva, amely nem engedélyezett vagy valami hiba lehetőség van benne, mint például, hogy hiányzik egy szorzás jel. Majd 2. lépésként ellenőrzöm, hogy a függvénynek van-e olyan pontja, ami a játékos bázisában van, ezzel együtt amennyiben a számítás hibát dob, akkor valamilyen elírási hiba történt. Ha minden jól ment, kiszámolja a függvény pontjait, és továbbítja minden játékosnak, majd a frontend megjeleníti, úgyszólván összeköti a pontokat.

Moderáláshoz van egy szuper admin, aki más felhasználókat tud „előléptetni” adminná. Minden felhasználó tudja egymást jelenteni, az adminok ezeket a jelentéseket kezelik, és amennyiben szükségesnek látják tilthatnak egy felhasználót, vagy chat hozzáférésüket korlátozhatják, és ezzel nem lesznek képesek többet a chat szobába írni. Az adminok pedig sima felhasználót tudnak alakítani egymásból – kivétel a szuper admin, ő semmilyen szempontból nem változtatható.

Felhasználók barát kapcsolatot képesek egymással létesíteni, ehhez a chat szobában kell a barát kérelem küldés gombra kattintani. Barátságoknak 2 státusza lehet, folyamatban, vagy ennek a tagadása. Tehát minden barátság létrejötte előtt el kell fogadni a kérést, és ezután tudnak egymással beszélgetni a felhasználók. Ahhoz, hogy a kommunikáció élő legyen WebSockets-t használok. Felhasználók továbbá le tudják egymást tiltani, ezzel soha többet nem kapnak a tiltott féltől üzenetet a chat szobákban.

Adatok tárolására nemcsak adatbázist használok, ugyanis vannak adatok, amik nem élnek sokáig, de jól jön, ha a programban bárhol gyorsan elérhető, és bár az adatbázissal bárhol elérhető, nem a legjobb, ha minden apró információért egy lekérést kell elindítani. Ehhez statikus változókat definiáltam, amiben tárolom az ilyen adatokat, játékok, online felhasználók stb.

7 Irodalomjegyzék

- Bekkhush, Simen. *Jest*. 06 05 2023. <https://jestjs.io/docs/getting-started> (accessed 05 26, 2023).
- Brian, Peiris. *Sequelize*. 2023. 05 25. <https://sequelize.org/docs/v6/getting-started/> (hozzáférés dátuma: 2023. 05 26).
- Hazel, Virdó, és Drake Mark. *How To Install MySQL on Ubuntu 20.04*. 2023. 05 18. <https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-20-04#step-1-installing-mysql>.
- Ismeretlen. *Angular*. n.d. <https://angular.io/docs> (accessed 05 26, 2023).
- . *Angular bootstrap*. n.d. <https://ng-bootstrap.github.io/#/home> (accessed 05 26, 2023).
- . *bootstrap*. n.d. <https://getbootstrap.com/docs/5.2/getting-started/introduction/> (accessed 05 26, 2023).
- . *Bootstrap.icons*. n.d. <https://icons.getbootstrap.com/> (accessed 05 26, 2023).
- . *datagrip*. 26 04 2023. <https://www.jetbrains.com/help/datagrip/creating-diagrams.html> (accessed 05 26, 2023).
- . *Express*. dátum nélk. <https://expressjs.com/en/4x/api.html> (hozzáférés dátuma: 2023. 05 26).
- . *Joi*. dátum nélk. <https://joi.dev/api/?v=17.9.1> (hozzáférés dátuma: 2023. 05 26).
- . *ngx socket io*. 2023. 05 11. <https://www.npmjs.com/package/ngx-socket-io> (hozzáférés dátuma: 2023. 05 26).
- . *npm jwt decode*. 2020. <https://www.npmjs.com/package/jwt-decode> (hozzáférés dátuma: 2023. 05 26).
- . *socket.io*. 2023. 05 02. <https://socket.io/docs/v4/> (hozzáférés dátuma: 2023. 05 26).
- . *supertest*. 2022. <https://www.npmjs.com/package/supertest> (hozzáférés dátuma: 2023. 05 26).
- . *typescript lang*. 2023. 05 26. <https://www.typescriptlang.org/docs/handbook/intro.html> (hozzáférés dátuma: 2023. 05 26).
- NodeJS. *Nodejs.org*. dátum nélk. <https://nodejs.org/en> (hozzáférés dátuma: 2023. 05 26).
- NodeSource. *distributions*. 2023. május 18. <https://github.com/nodesource/distributions#using-ubuntu-4>.
- Rudolf, Szendrei. *Segítség a pont és síkidom távolságának meghatározásához*. Rudolf, Szendrei, Budapest. 2021. 09 22.
- Srivatsan. *math.stackexchange*. 2011. 10 27. <https://math.stackexchange.com/questions/76457/check-if-a-point-is-within-an-ellipse#answer-76463> (hozzáférés dátuma: 2023. 05 26).

