Eötvös Lóránd Tudományegyetem

Informatikai Kar

Média- és Oktatásinformatika Tanszék

Online valós idejű torpedójáték

Heizlerné Bakonyi Viktória

Mester Tanár

Major Gábor

Programtervező informatikus Bsc

Budapest, 2018

Tartalom

Bevezetés

A témaválasztás indoklása

A webes világ engem mindig is lenyűgözött, különösen az tetszik benne, hogy a kódot egy böngészőben futtatva szemmel láthatóvá válik a munkánk. Szakmai gyakorlatom során frontend fejlesztéssel foglalkoztam, amitől csak még inkább kedvem lett az informatika ezen területével foglalkozni.

A téma választásakor számomra elsődleges szempont volt egy olyan program készítése, amelyet mások is szívesen használnának, ezért döntöttem a játékprogram készítése mellet.

A feladat leírása

Az általam megvalósított program egy online valós idejű torpedó játék, ahol a játékosok más, a hálózaton (akár Interneten) lévő játékosok ellen tudnak játszani. A játékhoz regisztráció szükséges, ahol a felhasználó a személyes adatai megadása mellet megadja a fiókhoz tartozó felhasználói nevet, ami egyedi és a többiek számára látható azonosító. Bejelentkezés után, amennyiben a játékra jelentkezik, a szerver megpróbál ellenfelet találni számára, ehhez legalább két játékra jelentkezett felhasználó kell. A program nyomon követi a játékra jelentkezettek számát és ezt az információt mindig megosztja a már jelentkezett játékosokkal, így a felhasználó láthatja, hogy vár-e még valaki rajta kívül a játékra.

A játék a szabályoknak megfelelően zajlik: a játékosok elhelyezik a hajóikat az erre kijelölt játéktéren és utána az ellenfél hajóinak pozícióit kitalálva, megpróbálják elsüllyeszteni egymás hajóit. A játék folyamatosságát időzítők biztosítják, így elkerülhető a fölösleges várakozás az ellenfélre. A csata addig folyik, amíg az egyik játékos el nem süllyeszti az ellenfél hajóit, vagy valaki kettőjük közül fel nem adja a játékot.

A csata után a felhasználóhoz tartozó statisztikák, a végeredmény függvényében frissülnek. Ezeket bárki akár bejelentkezés nélkül is megtekintheti az oldalon. Azért, hogy fölöslegesen hosszú listákat elkerüljük minden kategóriából (legtöbb: csata, győzelem, pont) csak az első húsz játékost jelenik meg.

Felhasználói dokumentáció

Célkitűzés

A program célja, hogy lehetővé tegye a népszerű torpedó játék játszását egy online felületen két játékos között, valós időben, mindezt úgy, hogy egyidejűleg a lehető legtöbb felhasználót tudja bevonni. Ennek megvalósítása érdekében a játékosok számára minél egyszerűbben és könnyebben kezelhető felületet kell biztosítani a lehető legjobb játékélmény eléréséhez.

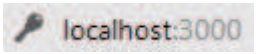
A programnak továbbá biztosítania kell, hogy a felhasználók tudjanak regisztrálni, be-, illetve kijelentkezni az oldalról. E mellet szeretnénk statisztikát is vezetni a felhasználókról. Az ehhez szükséges információkat és szolgáltatásokat, a host gépen futó programokkal kívánjuk ellátni úgy, hogy a felhasználóknak ne kelljen további programokat telepíteniük.

Rendszerkövetelmények a felhasználókhoz

A felhasználó részéről nincs szükség program telepítésére. Egy internet böngésző elegendő. Javasolt program: Chrome v.68 vagy egy újabb verzió. Természetesen feltételezzük, hogy a felhasználó eszköze azonos hálózaton van a host eszközzel.

A program használata

Ha a működéséhez szükséges szolgáltatásokat futnak a gépünkön, akkor a **localhost** 3000-es port-ján érjük el a programot. Ezt a böngésző url mezőjébe kell írni.



1. Ábra: az oldal url címe

A mai trendeknek megfelelően a weboldal reszponzív, ami azt jelenti, hogy az oldal alkalmazkodik az eszköz képernyő méretéhez. Erre azért van szükség mivel az oldal ugyanúgy látogatható okostelefonról, vagy tabletpc-ről, mint asztali számítógépről.

Menü

Az oldalak tetején található a menüsor, melynek jobb szélén találhatók az oldal bejárásához szükséges linkek. A menüpontok attól függően jelennek meg vagy tűnnek el a kijelzőn, hogy website melyik oldalán (login, signup, account, stb.) állunk, illetve, hogy bejelentkeztünk-e. Ha bejelentkeztünk, akkor a menüsorban láthatóvá válik, hogy milyen felhasználónévvel léptünk be. A bal oldalon található „Battleships game” feliratra kattintva vissza kerülhetünk a főoldalra.

2a. Ábra: fő oldali menü kijelentkezve

2b. Ábra: fő oldali menü bejelentkezve

Alacsony felbontású monitorokon és telefonokon a menü kompakt formában jelenik meg. Ekkor a menüpontokat a jobb szélen látható menü ikon megnyomásával érhetjük el.

3a. Ábra: menü kompakt nézetben, összecsukva

3b. Ábra: menü kompakt nézetben, kinyitva

Regisztráció

Regisztrálni külön oldalon lehet. Ezt az oldalt a „SignUp” menüpont meggnyomásával érhetjük el, ez csak akkor lehetséges, ha az oldalon nem vagyunk bejelentkezve egy másik felhasználóval. A regisztrációhoz szükséges megadnunk a kereszt-, és vezetéknevünket,

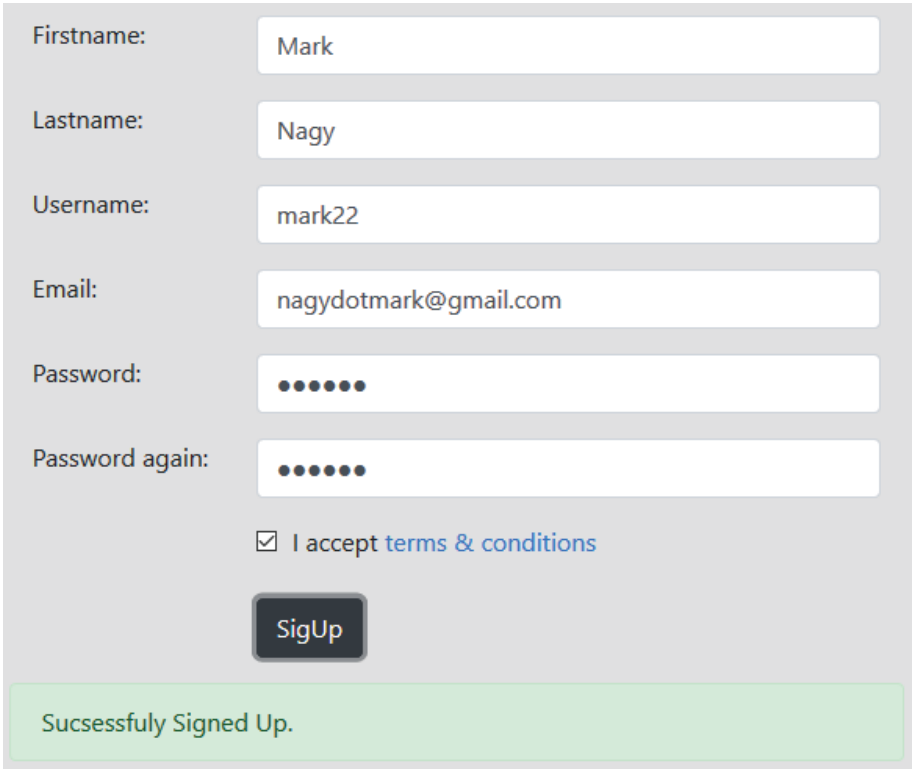
egy egyedi felhasználónevet, ami a továbbiakban már nem módosítható, egy email címet, illetve jelszót kell választanunk.

A beadott adatok ellenőrzését a szerver végzi. Hibás adat esetén a felhasználó hibaüzenetet kap, ami jó látható módon, a hibás beviteli mező alatt pirossal jelenik meg. Ilyen hiba lehet:

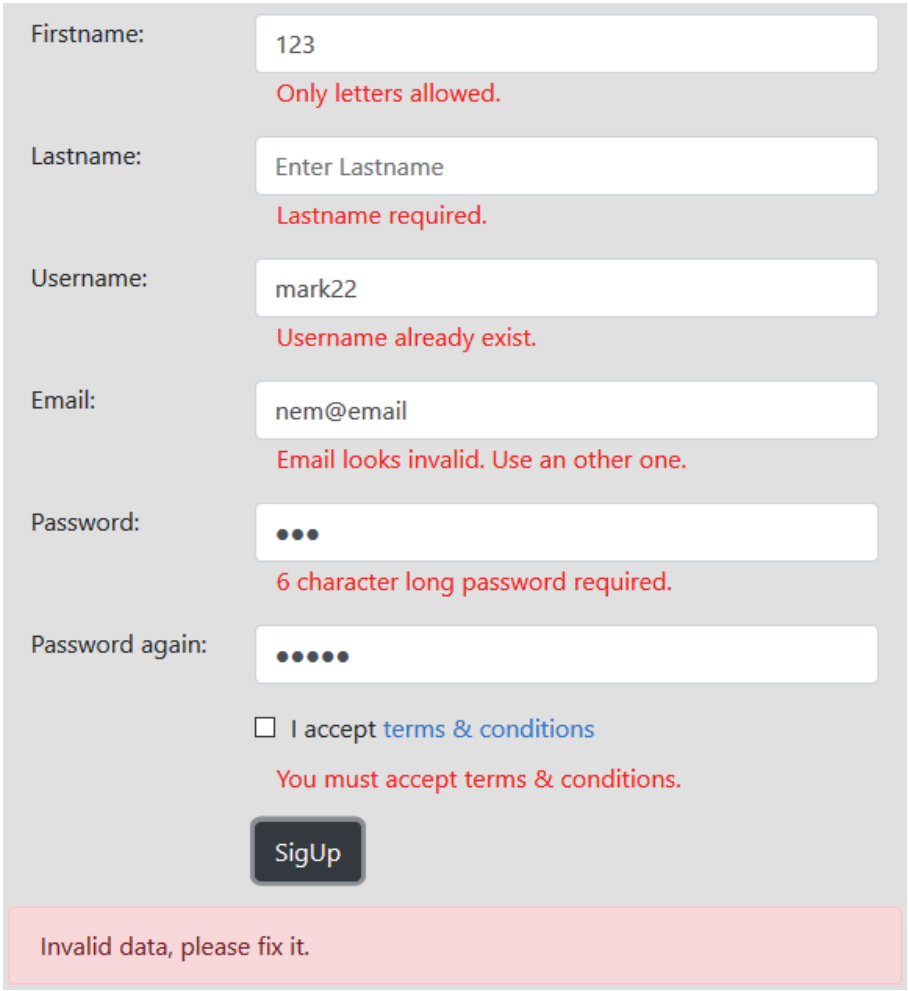
- ha egy mezőt üresen hagyunk,
- ha a kereszt-, és/vagy a vezetéknévben számot, szóközt vagy speciális karaktereket használunk (ide értve az ékezetes betűket is),
- ha nem egyedi, vagy legalább hat karakter hosszú felhasználónevet választunk,
- ha az e-mail cím nem megfelelő formátumú,
- ha az általunk megadott jelszó nem legalább hat karakter hosszúságú, vagy nem egyezik meg a második megadással
- ha nem fogadjuk el a végfelhasználói feltételeket

Ha a fenti hibák bármelyikét elkövetjük, akkor minden hibás mező alatt megjelenik a hibát kifejtő üzenet, illetve az űrlap alján megjelenő üzenet figyelmeztet minket, hogy ellenőrizzük a beírt adatokat. Sikeres regisztráció esetén egy zöld mező jelenik meg az űrlap alján, amely közli velünk a regisztráció sikerét. Ekkor egy kis idő elteltével a program átirányít minket a bejelentkezési oldalra.

#	Jó bemeneti adatok	Rossz bemeneti adatok
firstname:	Márk	123
lastname:	Nagy	*üres*
username:	mark22	mark22 (már foglalt)
email:	nagydotmark@gmail.com	nem@email
password:	123456	123
password agan:	123456	q2w3e



4a. Ábra: sikeres regisztráció, helyes adatokkal



4b. Ábra: sikertelen regisztráció, helytelen adatokkal

Bejelentkezés

A bejelentkezési oldalra a „LogIn” menüpont segítségével tudunk eljutni. A bejelentkezés a regisztrációhoz hasonló űrlapon történik. Itt a felhasználónevünket és a jelszavunkat kell megadunk. Hibaüzenetet akkor kapunk, ha ezt a két mezőt üresen hagyjuk, ha nem létező felhasználónevet, vagy ha rossz jelszót adtunk meg.

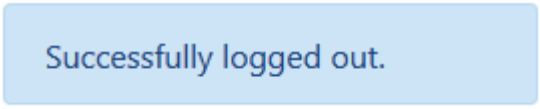
Lehetőségünk van arra, hogy az oldal emlékezzen ránk, így a böngésző bezárása és újra nyitása esetén nem kell újra bejelentkeznünk, (biztonsági okokból 30 naponta meg kell ismételni a bejelentkezést).

5. Ábra: az oldal emlékszik a bejelentkezett felhasználókra, ha kérik

Sikeres bejelentkezés esetén az űrlap alján erről értesít minket az oldal, majd kis idő elteltével átirányít minket a profilunkhoz.

Kijelentkezés

Kijelentkezni a menüsor jobb szélén lévő „Logout” link megnyomásával lehet. Sikeres kijelentkezés esetén az oldal átirányít minket a bejelentkező oldalra, ahol üzenetet kapunk a sikeres kilépésről. A „Logout” funkció a bejelentkezéskor bejelölt oldal emlékeztető parancsot is visszavonja, így a következő alkalommal, ismételten be kell jelentkeznünk a felhasználóneünk és a jelszavunk megadásával.



6. Ábra: a szerver üzenete sikeres kijelentkezéskor

Scoreboard és statisztika

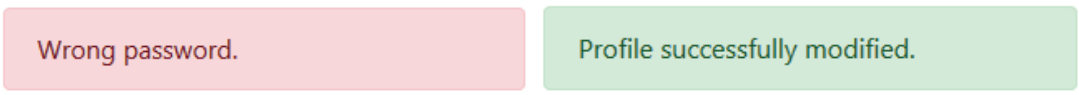
A játék során a program feljegyzéseket készít arról, hogy egy játékos hány játékot játszott, ebből mennyit nyert meg és mennyi pontot gyűjtött össze. Ezeket a felhasználók nyomon követhetik a profiljukban.

Az oldalon a „ScoreBoard” menüpont alatt érhető el az a lista, amely a kategória (legtöbb csata/győzelem/pont) húsz legjobb játékosának a felhasználó nevét és statisztikáját tünteti fel. Az adott kategória megtekintéséhez a táblázat fölötti menüből kell kiválasztani a megfelelő menüpontot. Ez a lista bárki számára elérhető, használata bejelentéseshez nem kötött.

Profil

A profilunkba belépve lehetőségünk van a regisztrációkor megadott adataink megtekintésére és módosítására (a felhasználónév nem módosítható). A profil oldal megtekintése bejelentkezéshez kötött. Bejelentkezés után az oldal automatikusan átirányít minket a profilunkba. A belépés után a fenti menüsoron megjelenik az „Account” menüpont. Ha bejelentkezés nélkül lépünk az oldalra, akkor az alkalmazás átirányít minket a bejelentkezési űrlaphoz.

Az oldalra belépve két kártyát láthatunk. Az alsó kártyán egy menürendszer segítségével tekinthetjük meg a felhasználónkhoz tartozó adatokat, vagy statisztikákat. Az adataink módosításához a „Modify” menüpontot kell kiválasztanunk. Itt egy regisztrációnál használt űrlaphoz hasonló űrlap fogad minket. Az űrlapnak csak azon mezőit kell kitöltenünk, amiket módosítani szeretnénk. Módosítási szándékunkat jelszavunk megadásával meg kell erősítenünk.



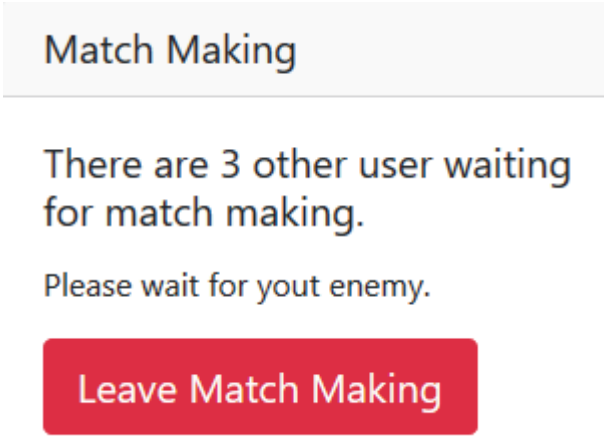
7. Ábra: az adat módosító űrlap üzenetei sikertelen illetve sikeres adatmódosításkor

A „Changepassword” menüpont alatt, a jelszó módosítására is lehetőségünk van. Itt adhatunk meg új jelszót a fiókunkhoz. A jelszónak, úgy mint a regisztrációkor itt is legalább hat karakter hosszúnak kell lennie és szintén kétszer kell megadnunk. Természetesen ezt is meg kell erősítenünk a jelenleg használt jelszavunkkal.

Játék indítása

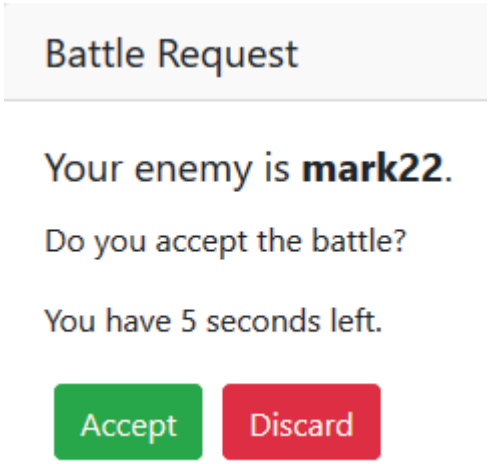
A játékot bejelentkezés után, a profilunkból a „Start Game” gomb megnyomásával tudjuk indítani. Ekkor közvetlen a „match making” folyamatba kerülünk be, melynek lényege, hogy a szerver keres számunkra egy másik játékost, aki szintén jelentkezett játékra, tulajdonképpen ebben a fázisban egy ellenfelet keres nekünk.

Miközben várunk az ellenfelünkre nyomon követhetjük, hogy rajtunk kívül még mennyi játékos vár a sorsolásra. Ez az információ folyamatosan frissül, így nyomon követhetjük ha valaki csatlakozik, vagy távozik a „match making” folyamatból. Távozni a piros „Leave Match Making” gomb megnyomásával lehet.



8. Ábra: várakozás az ellenfél sorsolásra

Legalább két „match making”-re jelentkezett játékos szükséges ahhoz, hogy a program párba tudja állítani őket. Ekkor a játékosok értesítést kapnak arról, hogy kit sorsoltak ki számára ellenfélként. A játék akkor kezdődik, ha a játékosok kölcsönösen elfogadják egymást, mint ellenfél. Elfogadni a zöld „Accept” gomb megnyomásával lehet, erre tíz másodperce van a játékosoknak. Természetesen a piros „Discard” gomb megnyomásával lehetőség van az ellenfél elutasítására is. Ekkor az a játékos aki elutasította az ellenfelét visszakerül a „match making” állapotba, hogy a szerver új ellenfelet sorsolhasson számára. Az a játékos, akit valaki nem fogadott el ellenfélként, értesítést kap a visszautasításról, majd ő is visszakerül a „match making” állapotba.



9. Ábra: Az ellenfél elfogadása vagy elutasítása

Az időlimit jelzés jól látható helyen a gombok felett helyezkedik el. Ha a rendelkezésre álló idő letelt, de a játékos nem nyomott gombot, akkor az automatikusan az ellenfél elutasítását jelenti („Discard”).

Játékra csak olyan játékos jelentkezhet, aki se „match making” állapotban, se játékban nincsen. Ellenkező esetben a gomb megnyomásakor üzenetet kapunk arról, hogy nem kerültünk „match making” állapotba, mivel az előző feltételeknek egyikének nem felelünk meg.

Játék menete

A játék az ellenfelek kölcsönös elfogadása után indul. Erre az oldalra csak az elfogadást követő lépésben kerülhetünk át. A játéktér két nagyobb egységből és a felettük lévő információs sávból áll. Ezen az információs sávon láthatjuk ellenfelünk felhasználónevét, illetve itt található egy „Exit Game” gomb, melynek megnyomásával bármikor véget vethetünk a játéknak, magyarán feladhatjuk azt és visszatérhetünk a profilunkhoz.

Az információs sáv alatt találjuk a játékmezőt. Itt a „Your Ships” nevű tízszer tízes mezőn kell elhelyeznünk a hajóinkat. Egy öt, egy négy, egy három, két kettő és két egy egység hosszú hajót, ebben a sorrendben. A szabályoknak megfelelően hajóinkat függőlegesen és vízszintesen is elhelyezhetjük, viszont nagyon fontos, hogy a hajók egymást nem fedhetik, a játék mezőről ki nem lóghatnak és az oldalaik mentén nem érhetnek össze.

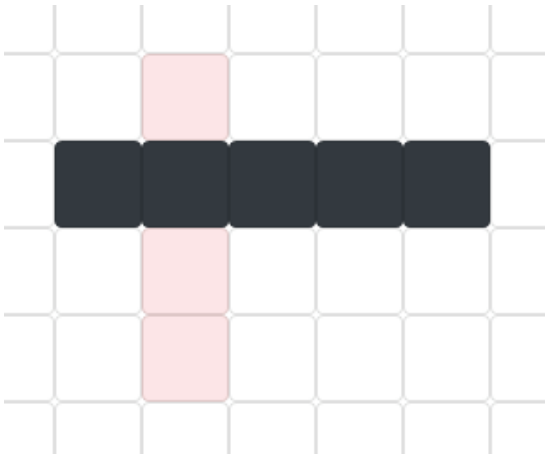
A program a hajók elhelyezését úgy segíti, hogy halványan megjelöli azokat a mezőket, amelyre a hajó kerülni fog. A hajót az egér bal gombjának kattintásával tudjuk elhelyezni. Az elhelyezett hajókat a program sötét szürke színnel jelzi. Ha az irányt szeretnénk

módosítani, azt a **szóköz** billentyű lenyomásával tehetjük meg (miközben a kurzort a mező fölött tartjuk).

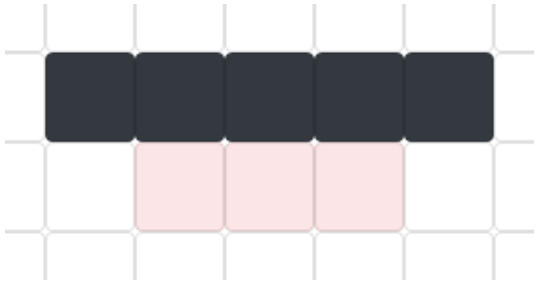
10a. Ábra: a hajó leendő helye

10b. Ábra: lehelyezett hajó

Ha egy hajót nem tudunk lehelyezni, mert azzal megsértenénk a játékszabályokat, akkor ezt a program halvány pirossal jelzi. Ekkor hiába kattintunk az egérrel nem kerül lehelyezésre hajó.



11a. Ábra: a hajók nem lóghatnak egymásra



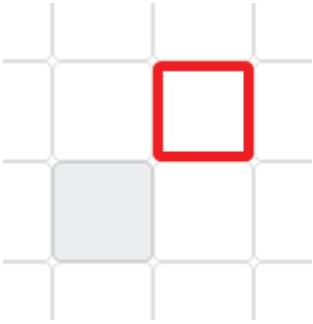
11b. Ábra: a hajók nem érintkezhetnek az oldalaikon

A játékmező alatt három gomb található. Ezek sorra a „Reset Table”, a „Ready” és a „Random” gombok. Az első gomb addig nem kattintható, amíg nincs elhelyezve legalább egy hajó a mezőn. Ezen gomb megnyomásával a játékmezőről eltűnnek az eddig elhelyezett hajók, így újra tudjuk kezdeni a lerakást. Fontos megjegyezni, hogy a korrekció elvégzése a játékos rendelkezésére álló időt nem állítja meg, vagy tekeri vissza. A második gomb akkor válik kattinthatóvá, ha az összes hajót a szabályoknak megfelelően elhelyeztük. Ennek a gombnak a megnyomásával véglegesíthetjük a hajóink pozícióját, megnyomásával ezek a gombok eltűnnek és leáll a visszaszámláló funkció is. A „Random” gomb megnyomásával a program véletlenszerűen helyezi el számunkra a hajókat.

A hajók elhelyezésére negyvenöt másodperc áll rendelkezésünkre. Ha végeztünk a hajók letételével nyomjuk meg a „Ready” gombot. Ha ezt elmulasztjuk, akkor az idő letelte után a játéknak ugyanúgy vége szakad, mint ha megnyomtuk volna az „Exit Game” gombot.

Amikor mindkét játékos jelezte, hogy készen van a hajók elhelyezésével és rákattintott a „Ready”-re, akkor a program véletlenszerűen kisorsolja azt a játékost, aki először kezdheti a torpedók kilövését. Ekkor már mindkét játékos számára megjelenik a „Enemy Area” nevű játéktér, majd az információs sávban a játékosok láthatják, az „Enemy turn. Please wait”, vagy a „You turn. Fire a missile on the Enemy Area” üzenetet, mely egyértelműen jelzi, hogy melyik játékos mikor következik.

A soron következő játékos az „Enemy Area” nevű játéktéren kiválaszthatja, hogy hova szeretne torpedót indítani. Egy körben csak egy mezőt választhat. Kiválasztani az egér bal gombjának kattintásával lehet, ekkor a mező körüli piros szegély jelenik meg, ami egyértelműen jelzi a választásunkat. A program itt is segít azzal, hogy azt a mezőt, amelyre a kurzorunk mutat, halványan megszínezi.



12. Ábra: a pirossal megjelölt mező a torpedó célpontja

A kiválasztott célpontot mindaddig lehetőségünk van módosítani, amíg meg nem nyomtuk a játéktér alatti található „Fire” gombot. A gomb csak akkor kattintható, ha van kijelölve mező, és csak annak a játékosnak jelenik meg a kijelzőjén, aki éppen következik. A játék folyamatosságának biztosítása érdekében a játékosoknak tizenöt másodpercünk van a torpedó célpontját kiválasztani és megnyomni a „Fire” gombot. Ellenkező esetben a játék vége szakad.

Amennyiben a „Fire” gombra kattintunk az az aktuális kör végét jelenti, ekkor az ellenfelünk következik. Arról, hogy a torpedó eltalálta-e az ellenfél hajóját azonnal üzenetet kapunk. Az üzenet az „Enemy Area” alatt ott jelenik meg, ahol eddig az időzítőt

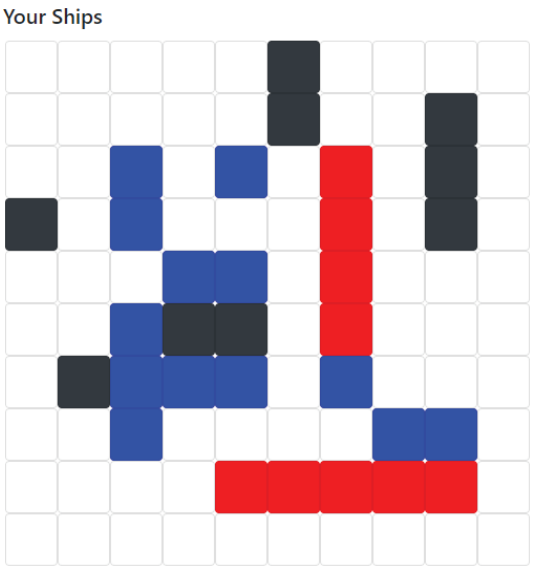
láttuk. Természetesen az ellenfél is értesül a lövés kimenetéről. Ő a „Your Ships” alatt található mezőben olvashatja ezt. A következő üzenetek jelenhetnek meg:

- „You MISSED the enemy ship.”
- „You HIT the enemy ship.”
- „You SANK the enemy ship.”
- „The enemy MISSED your ship.”
- „The enemy HIT your ship.”
- „The enemy SANK your ship.”

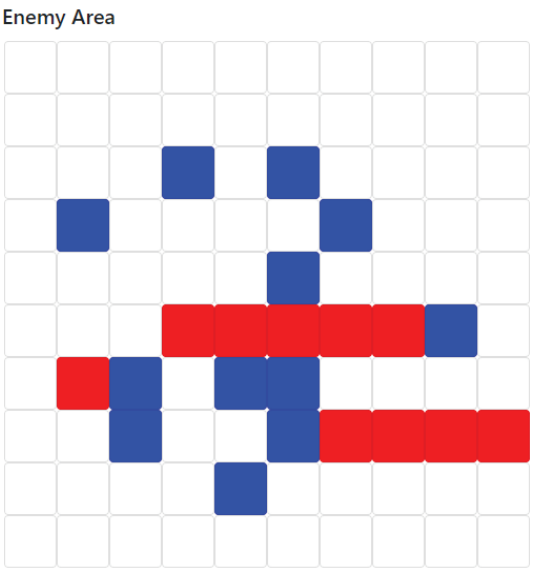
Az üzeneteken kívül a lövés eredménye természetesen a játék mezőn is megjelölésre kerül, attól függő színnel, hogy a torpedó talált-e vagy sem. Ha talált, akkor piros, ha nem akkor azt kék színnel jelzi a program. A játékosok saját hajóit érő torpedók a „Your Ships” mezőkön, az általuk kilőtt torpedók pedig az „Enemy Area” mezőkön jelennek meg.

Az alábbi ábrán a játék egy előrehaladott állapota látható. Ha megfigyeljük az **A játékos** által kilőtt torpedókat (lásd **A játékos** „Enemy Area”) és a **B játékos** hajóit tartalmazó játékteret (lásd **B játékos** „Your Ships”), akkor jól láthatjuk, hogy a program milyen módon jelzi a torpedó lövések eredményét.

A játékos

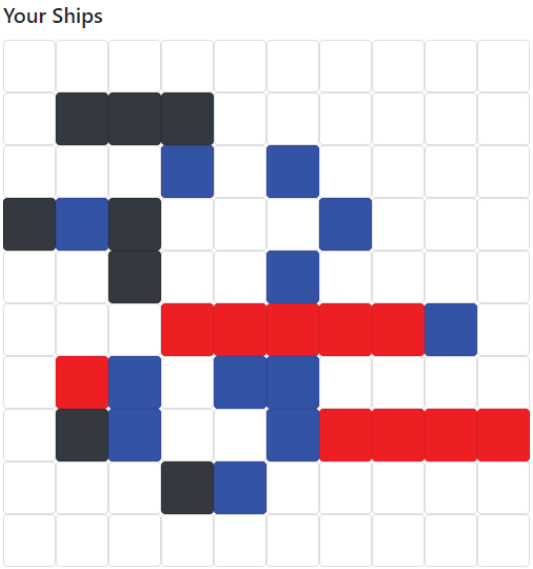


13.a Ábra: az A játékos hajóinak táblája

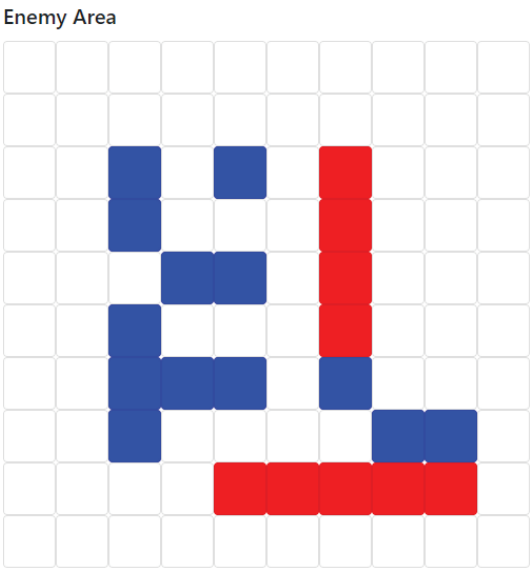


14.a Ábra: az A játékos által kilőtt torpedók táblája

B játékos



13.b Ábra: a B játékos hajóinak táblája



14.b Ábra: a B játékos által kilőtt torpedók táblája

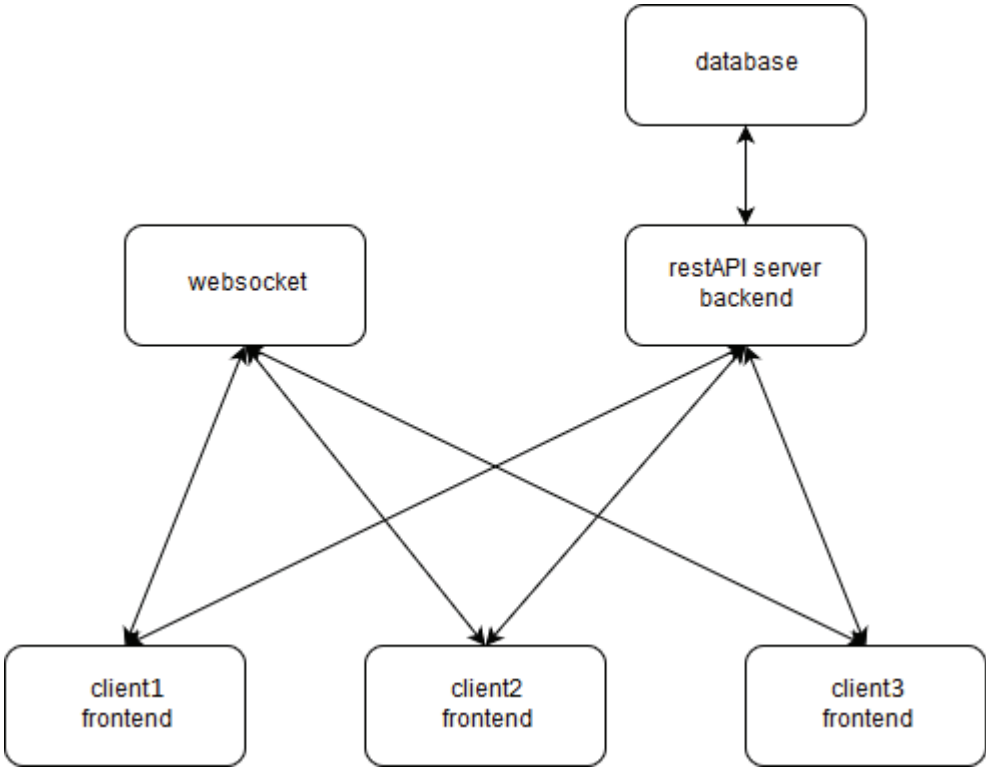
A játék végeztével a győztest harminc, ellenfelét tíz ponttal jutalmazza a program. Ezután a profil oldalunkról akár rögtön jelentkezhetünk egy újbóli játékra.

Fejlesztői dokumentáció

Specifikáció

Architektúra

A program négy nagyobb önálló egységből tevődik össze. Az első kettő a szerver, más néven a backend és az adatbázis. Előbbi egy PHP restAPI alkalmazás, ami a végpontokon keresztül biztosítja a kommunikációt az adatbázissal. Ezektől jól elkülöníthető a frontend, ami egy React alkalmazás és a valós idejű játékélményért felelős websocket.



15.Ábra: a program architektúrája

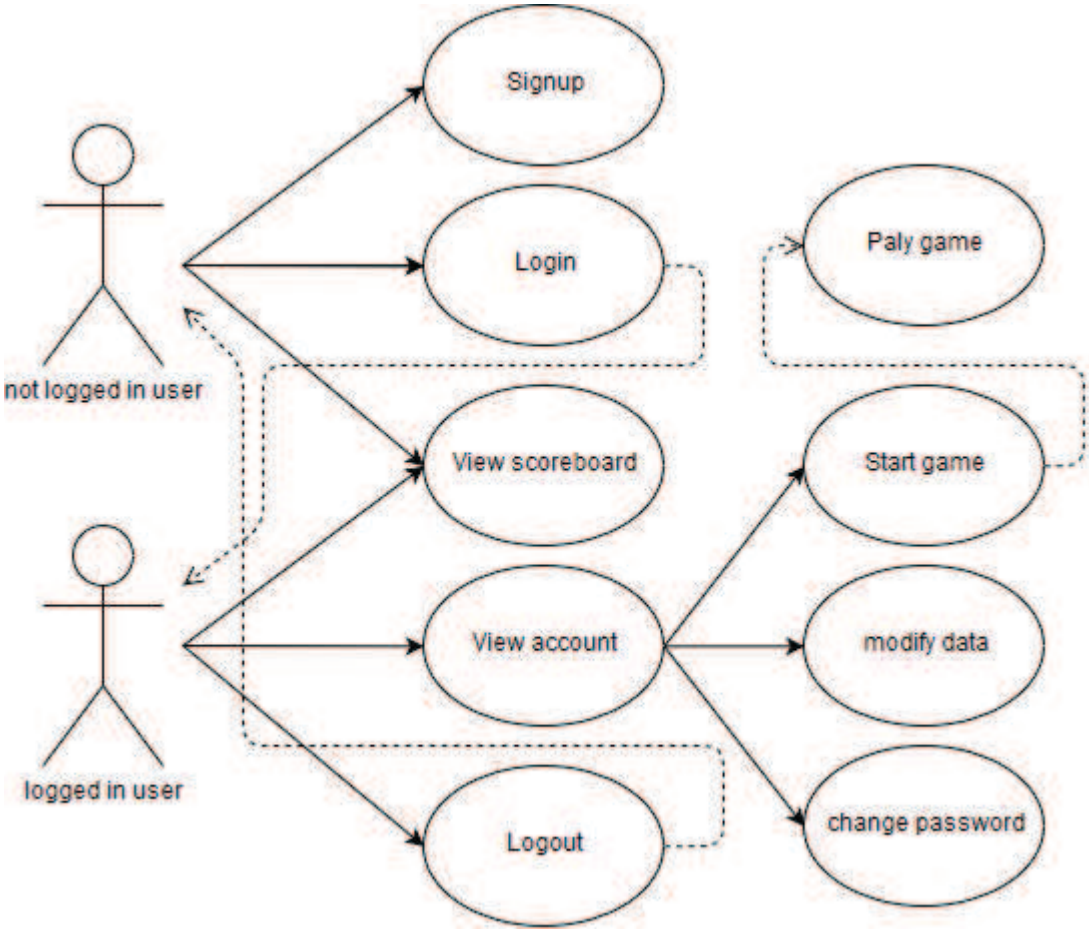
Use-case diagramm

Az oldalra látogató felhasználókat két nagy csoportba soroljuk aszerint, hogy be vannak-e jelentkezte, vagy sem. Ennek megfelelően van lehetőségük a különböző szolgáltatások használatára.

A be nem jelentkezett felhasználó számára elérhető funkció a regisztrálás és a bejelentkezés. Utóbbi funkciót használva átkerülnek a bejelentkezett felhasználók közé.

Bejelentkezett felhasználók számára az előbbi két funkció használata nem lehetséges, mivel a program szempontjából nincs értelme. Csak számukra elérhető a felhasználói fiók, ahol tudják módosítani adataikat, tudnak jelszót cserélni és innen tudják elindítani a játékot. A kijelentkezés funkció segítségéve visszakerülhetnek a be nem jelentkezett csoportba.

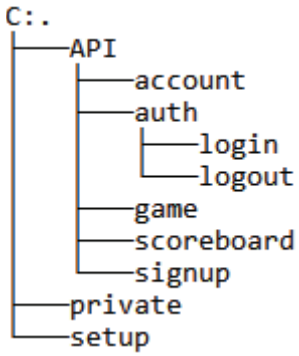
A rangsor táblát mindkét csoport megtekintheti.



16. Ábra: a programhoz tartozó use-case diagramm

Program könyvtár struktúrája

A backend végpontjait tartalmazó fájlok az **API** mappában találhatóak. A **private** mappában olyan állományokat találunk, amelyek elengedhetetlenek a szerver működéséhez. Itt találjuk a bejelentkezést vizsgáló és az adatbázist kezelő logikákat. Az adatbázist és a táblát elkészítő szkripteket a **setup** mappában találjuk.

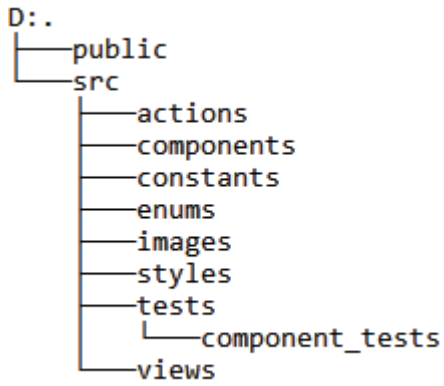


17.ábra: a backend könyvtár struktúrája

A frontend alkalmazás a **create-react-app** csomag felhasználásával készült el. Ez nagyban befolyásolja könyvtár végső szerkezetét. A **public** mappában találjuk az index.html file-t, amibe mindig az aktuális **routing**-nak megfelelő komponenseket láthatjuk. Az **src** könyvtár tartalmazza a feladat megoldását. Itt helyezkedik el az index.js ami az általunk írt JavaScript kódokat belehelyezi az index.html-be, és az App.js ami a **routing** működéséért felelős és persze itt található az oldal stílusát leíró App.css is.

Az alábbi könyvtárak az **src**-n belül találhatók.

- Az **actions** mappa, amiben az alkalmazás számára fontos függvényeket és a játék logikát tároljuk.
- Az oldalt felépítő komponenseket, melyeket a **components** könyvtárban találunk.
- A **constants** mappában találjuk azokat a leíró fájlokat, amik az űrlapok generálásához szükségesek, illetve ide vettük fel a szerver végpontjainak és az oldalak url címét.
- A program jobb átláthatósága érdekében vettünk fel enum típusokat. Ezeket az **enums** mappában hoztuk létre.
- Az oldalon felhasznált képeket és animációkat az **images** mappába helyeztük.
- A **styles** mappa azokat az scss állományokat tartalmazza, amelyek meghatározzák az oldal stílusát. A css generálása miatt fontos hogy ezeket a fájlokat úgy nevezzük el, hogy nevük előtt egy alsóvonás szerepeljen.
- Az automatizált tesztek a **tests** könyvtáron belül találhatóak. A komponensek tesztjeit ezen belül összegyűjtöttük a **component_tests** mappába.
- A **views** mappában helyezkednek el az oldal úgymond fő komponensei. A routing során ezeket a komponenseket tölti be az oldal. Az ő feladatuk összefogni és felparaméterezni az oldalon megjelenő összes komponenst.



18. Ábra: a frontend könyvtár struktúrája

Alkalmazott technológiák

PHP[1]

A PHP másnéven Hypertext Preprocessor egy szerveroldalon futó platformfüggetlen szkriptnyelv, melynek segítségével egyszerűen készíthetünk dinamikus és interaktív weboldalakat. Nagy népszerűségnek örved a webes világban, a legnépszerűbb és leglátogatottabb oldalaknál is lehet vele találkozni, ilyen például a Google.com, Facebbok.com vagy a WordPress.com. Erősségeihez tartozik, hogy a beépített függvényeinek segítségével:

- képes a **cookie** küldésére és fogadására,
- tudja kezelni a **http response** kódokat és a **Json** fájlokat, így egyszerűen készíthető vele **restAPI** alkalmazás,
- a legtöbb adatbázishoz procedurálisan vagy akár objektum orientáltan is képes kapcsolódni, például: MySQL, Oracle, stb.

A PHP scripteket `<php` és `?>` határoló tagek közé írva, akár egy html fájlba is beletehetjük őket, csak ekkor arra figyeljünk, hogy az állomány kiterjesztését változtassuk meg .php-re. A kód interpretálása szerveroldalon történik mielőtt a webszerver visszaküldené a válaszát a kliensnek. A PHP szkriptek végrehajtása után a kimenet egy html fájl, így a kliensoldalon lévő felhasználók számára a kód nem látható. Általánosságban elmondható, hogy az Apace webszerverrel használják a legtöbben. Legfőbb kihívója az ASP.

MySQL[2]

A MySQL adatbázis a lekedveltebb a PHP fejlesztők körében. Gyors, megbízható, ingyenes és egyszerű használni. Egyaránt alkalmas kisebb vagy nagyobb alkalmazások kiszolgálására. Sintaxisát tekintve sztenderd SQL-t használ. Szintén népszerű a legnagyobbak között, mint a Youtube.com, a Facebook.com, vagy a Twitter.com.

Node.js és npm[3]

A Node.js platformfüggetlen, nyílt forráskódú szerver alkalmazás, ami JavaScript-et futtat szerver oldalon. Többek közt használhatjuk:

- dinamikus oldalak létrehozására,
- fájl műveletek végrehajtására a szerver gépen,
- adatok gyűjtésére űrlapokról
- adatok hozzáadásához, módosításához vagy törléséhez egy adatbázisban

A kódot a **Chrome's V8 JavaScript engine** fordítja át gépi kódra, ami egy alacsonyabb szintű kódot eredményez. Az ilyen kódot a gép interpretálás nélkül képes futtatni, ami gyorsabb működést eredményez. Előnye a többi szerver alkalmazáshoz képest, hogy a fájl műveleteket aszinkron módon végzi, így elkerülhető, hogy a program blokkolódjon a fájl művelet időtartama alatt.

Az npm egy csomag menedzser a Node.js-hez. Segítségével egyszerűen adhatunk hozzá csomagokat az alkalmazásunkhoz, vagy oszthatjuk meg az álltunk készítetteket másokkal. Átláthatóvá és könnyen kezelhetővé teszi az általunk használt csomagok verzióinak a kezelését. Több mint hatszáz ezer JavaScript csomagot tartalmaz, ezekből csomagokból hetente megközelítőleg három milliárdot töltenek le. [4]

Fejlesztői környezet

Rendszerkövetelmények

Az alkalmazás nem operációs rendszer specifikus. A futtatáshoz szükséges programok a PHP v.7.2.2, MySQL v.10.1.30-MariDB, Node.js v.8.11.1 és npm v.5.6.0. Nem kizárt, hogy korábbi verziókkal is működne a program, de az elérhető legfrissebb szoftverek használata ajánlott.

A program működéséhez aktív internet kapcsolat szükséges. Telepítéskor a node-modulok letöltéséhez, azt követőleg pedig a széleskörű elérhetőséghez. A minél felhasználó barátibb eléréshez ajánlott domain cím használata (pl: www.torpedojatek.hu).

A program nem igényel különleges hardvereket, mint például dedikált videókártya, vagy hangkártya.

Backend

A backend működéséhez szükség lesz a PHP és MySQL programokra. Ezen programokat legegyszerűbben a XAMPP[5] Apache ingyenes disztribúció letöltésével biztosíthatjuk.

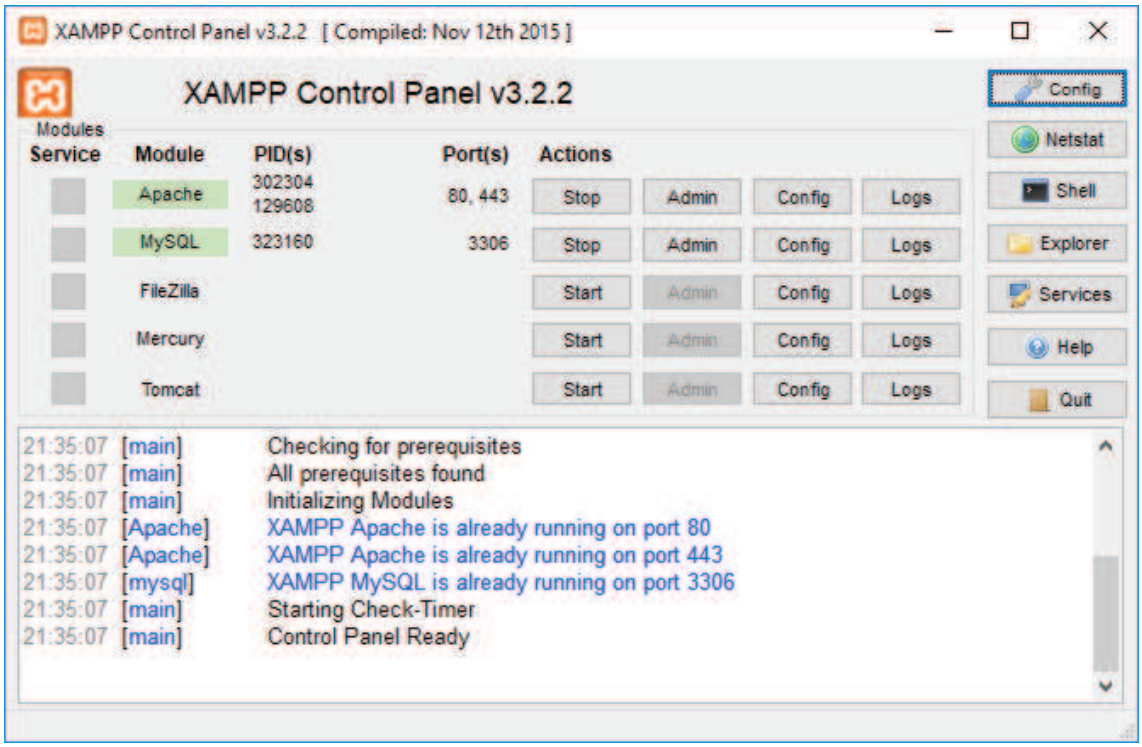
Töltsük le az oldalról az operációs rendszerünknek megfelelő installáló programot, majd kövessük a program utasításait. Vigyázzunk, hogy a program telepítése során a megfelelő komponensek letöltésre kerüljenek, úgy mit az Apache, MySQL, PHP, és phpMyAdmin.

Windows esetén ha mást nem állítottunk be akkor a program alapértelmezetten a **C:\xammpp** mappába települ. Ezen mappán belül találunk egy **htdocs** nevű mappát. Az ebben a mappában lévő fájlokat és könyvtárakat az Apache szerver indítása után a **localhost 80**-as port-ján érjük el. Ebbe a mappába kell elhelyeznünk a forrás fájlokat.

Az adatbázist elkészítő PHP **fájlokat** parancssorból kell futtatni, de ehhez előbb szükséges a PHP **path** beállítása.

Windows 10 esetén nyomjuk meg a windows billentyűt (⊞) és kezdjük el gépelni a „környezeti változók” szót. A megjelenő listáról válaszuk ki az „A rendszer környezeti változóinak módosítása” opciót. Ezt követően a felugró ablakban válasszuk ki a „Speciális” fület, majd kattintsunk a „Környezeti változók…” gombra. Itt válaszuk ki a „Path” nevű változót és nyomjunk a „Szerkesztés…” gombra. Adjuk hozzá a Path-hoz a **C:\xamppp\php** útvonalat, majd minden megnyitott ablakot az „OK” gomb segítségével zárjunk be. Ha volt nyitva parancssorunk vagy PowerShell ablakunk, akkor zárjuk be, és indítsuk újra őket.

A további műveletekhez el kell indítanunk az Apache és MySQL szolgáltatásokat. Ehhez nyissuk meg a XAMPP-ot (**xampp.exe**) és a „Start” gomb megnyomásával indítsuk el az előbbieket. Figyeljük meg, hogy az Apache a 80-as, míg a MySQL a 3306-os port-on működik. Vigyázzunk arra, hogy ne akadjanak össze más szolgáltatásokkal a gépünkön.



19. Ábra: a XAMPP kezelő felülete

Arról hogy a környezetet jól állítottuk fel és megfelelően működik, úgy győződhetünk meg, ha parancssort, vagy PowerShell ablakot nyitunk és begépeljük a **php -v** illetve **mysql -v** parancsokat. Ezek a parancsok megmondják, hogy az adott programból hányas verzióval rendelkezünk. Ha valami folytán nem sikerült volna megfelelően telepítenünk, vagy beállítanunk valamelyik programot, akkor hibaüzenetet kapunk. Az utolsó parancs után egy ctrl + C billentyű kombináció beütése szükséges. Ha minden sikerült az alábbit kell látnunk:

```
C:\Users\Gábor>php -v
PHP 7.2.2 (cli) (built: Jan 31 2018 19:51:55) ( ZTS MSVC15 (Visual C++ 2017) x86 )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies

C:\Users\Gábor>mysql -v
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 372
Server version: 10.1.30-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> Bye

C:\Users\Gábor>
```

20. Ábra: a PHP és MyAQL programok verzió számainak vizsgálata

A következő lépés az adatbázis létrehozása és beállítása. Ha a saját adatbázisunkat szeretnénk használni, akkor a szkriptekben írjuk át az adatbázis címét, a hozzátartozó felhasználónevet és jelszót.

A **htdocs** mappán belül a forrás fájlok között lépjünk be a **back-end** nevű mappába. Itt található a PHP szerver forráskódja. A **back-end\setup** mappában nyissunk meg a parancssort és futtassuk a **make-db.php** fájlt (**php make-db.php**), ez elkészíti számunkra a működéshez szükséges adatbázist. Ezt követően járjunk el hasonlóan a **create-table.php** fájllal, ami az előbb létrejött adatbázisban elkészíti a „**user**” nevű táblát, amiben a felhasználóink adatait fogjuk tároljuk. Sikeres végrehajtás esetén a következőt kell látnunk:

```
C:\xampp\htdocs\szakdolgozat\back-end\setup>php make-db.php
Database created successfully
C:\xampp\htdocs\szakdolgozat\back-end\setup>php create-table.php
Table 'users' created successfully
C:\xampp\htdocs\szakdolgozat\back-end\setup>
```

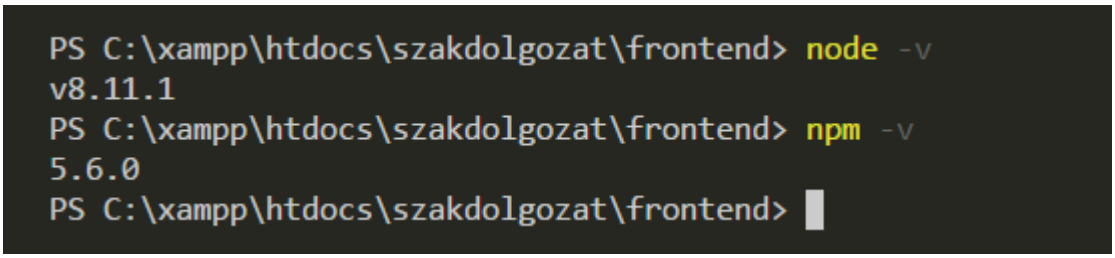
21. Ábra: az adatbázis elkészítő PHP szkriptek futtatása

Az adatbázis sikeres létrehozásáról úgy győződhetőnk meg, hogy egy böngészőt nyitunk, majd fent az url mezőbe a **localhost:80/phpmyadmin/** címet írjuk. Itt a phpMyAdmin felületén nyomon követhetjük és szerkeszthetjük az adatbázisunkat, bár az utóbbi művelet NEM ajánlott.

Frontend és Websocket

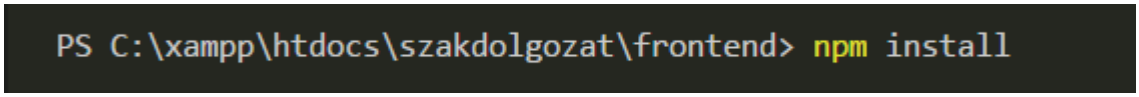
A frontend telepítéséhez elsőként le kell töltenünk a Node.js, ezt a programot a https://nodejs.org oldalról tudjuk ingyenesen beszerezni. Válasszuk a „Recommended For Most Users” opciót, és töltsük le.

A telepítés során kövessük az installáló szoftver utasításait és ne változtassuk meg az alapértelmezett beállításokat. A telepítés végeztével a helyes működésről, úgy győződhetünk meg, ha nyitunk egy parancssort, vagy PowerShell ablakot és begépeljük a **node -v** és **npm -v** parancsokat. Ha minden rendben van, akkor az alábbit kell látnunk:

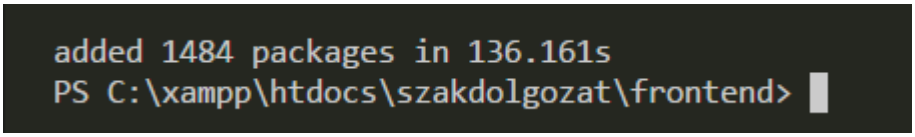


22. Ábra: a Node.js és npm programok verzió számainak vizsgálata

Siker esetén installálhatjuk az npm csomagokat. A **frontend** mappába nyissunk PowerShell ablakot és futtassuk az **npm install** parancsot. Ez telepíti a **node_modules** mappát, amiben a frontend és a websocket működéséhez szükséges könyvtárak találhatóak.



23.a Ábra: az npm csomagok installálása



23.b Ábra: az npm csomagok sikeresen települtek

Ha nincs hibaüzenet, akkor az installáció sikeresen befejeződött. A frontend elindításához PowerShell ablakban adjuk ki az **npm start** parancsot, ekkor az alapértelmezett böngésző megnyílik a **localhost** 3000-es port-on, ahol elérhetővé válik a játék weboldala.

Backend

Az adatbázis

A program a PHP által biztosított MySQLi függvényeket használja, hogy a MySQL adatbázishoz kapcsolódjon. Így egyaránt lehetőségünk van az objektum orientált és a procedurális használatra.

A **battleships** nevű adatbázisban található az **users** tábla. Ebben tároljuk a felhasználóink bejelentkezéshez szükséges adatait és a hozzájuk tartozó statisztikáit. A táblához kilenc lekérdezés tartozik, melyeket helyenként többször is felhasznál a program.

users
id: INT
firstname: VARCHAR (255) NOT NULL
lastname: VARCHAR (255) NOT NULL
usename: VARCHAR (255) NOT NULL UNIQUE
email: VARCHAR (255) NOT NULL
password: VARCHAR (255) NOT NULL
code: VARCHAR (255) NOT NULL
battles: INT
wins: INT
points: INT

A lekérdezések közül egyedül azt az egyet emelném ki, amelyik a rangsor tábla elkészítéséhez szükséges adatokat kérdezi le. A feladata egy adott szempont (csaták, győzelmek, vagy pontok) alapján vissza adni a húsz legjobb felhasználónevét és statisztikáját.

```
"SELECT username, battles, wins, points FROM  users ORDER BY ".$order." DESC LIMIT 20";
```

24. Ábra: a rangsor tábla elkészítéséhez használt SQL lekérdezés

Authentikáció

A bejelentkezés és annak a nyomon követése a PHP **session** segítségével a szerveren történik. Mivel minden kliens egyedi sessionid-val rendelkezik ezért ha valaki be van jelentkezve, akkor a klienshez tartozó felhasználónevet a **session[logged_in]**-ben tudjuk tárolni, kijelentkezett állapotban ennek értéke **null**.

A bejelentkezéskor meg lehet adni, hogy az oldal emlékezzen ránk akkor is, ha bezárjuk a böngészőt. Viszont a következő böngésző nyitáskor megváltozik a kliens sessionid-ja, ezért a **session**-ből már nem tudjuk kiolvasni, hogy valaki be van-e jelentkezve. Ennek kiküszöbölésére az oldal **cookie**-t használ, amit a bejelentkezéskor helyez el, ha bepipáltuk a „Keep me login” mezőt.

Működésben ez annyit tesz, hogy az oldal betöltésekor küld egy **GET** metódusú kérést az oldalhoz tartozó végponthoz, ahol az **auth** (bejelentkezést vizsgáló) függvény eredményétől függően kap választ az oldal.

Az **auth()** feltételezi, hogy a felhasználó nincs bejelentkezve, ezért először megnézi, hogy a **session[logged_in]**-ben talál-e felhasználót, ha igen, akkor jelzi, hogy valaki be van jelentkezve. Ha a **session[logged_in]** üres volt, akkor még megvizsgálja a kliens gépén lévő **cookie[remember]**-t, ha megtalálja a lehelyezett cookie file-t és a benne lévő adatok érvényesek, akkor jelzi, hogy valaki be van jelentkezve egyébként úgy értékeli, hogy a kliens nincs bejelentkezve.

```php
function auth() {
    if(isset($_SESSION["logged_in"])) {
        return true;
    }
    if(isset($_COOKIE["remember"])) {
        $cookie = unserialize($_COOKIE["remember"]);
        if( password_verify(
            get_code_for_remember($cookie["username"]),
            $cookie["code"]) ) {

            $_SESSION["logged_in"] = $cookie["username"];
            return true;
        }
    }
    return false;
}
```

*25. Ábra: az authentikációt végző **auth** föggvény*

Flash[6]

A backend-en a végpontoknál lehetőségünk van a kliensek számára olyan üzenetek készítésére, melyeket a szerver egy más végpontján tudunk elküldeni. Ilyen üzeneteket olyan esetben készítünk, amikor tudjuk, hogy a válasz elküldése után a frontend átirányítja a felhasználót. Tipikusan ilyen eset a kijelentkezés, ahol az API/auth/logout - GET végpont készít egy üzenetet, amelyet az API/auth/login - GET fog kiolvasni és elküldeni a kliensnek.

```php
save_to_flash([
    "message" => "Successfully logged out."
]);
```

26. Ábra: üzenet elkészítése

```php
$flashData = load_from_flash();
$flash["message"] = $flashData["message"] ? : NULL;
```

27. Ábra: üzenet kiolvasása

Védelem[7]

Elkerülendő, hogy a rosszindulatú felhasználók hozzáférjenek az adatbázishoz vagy kárt tegyenek a szerver működésében az űrlapokon beérkező adatokat ellenőrizni kell. Ennek érdekében készült a **defence** függvény, melynek segítségével minden, a kliens által küldött mező értékét ártalmatlaníthatjuk.

- elötte: <script>alert ('hacked ')</script>
- utánna: <script>alert('hacked)</script>

```php
function defender($str){
    $str = trim($str);
    $str = stripslashes($str);
    $str = htmlspecialchars($str);
    return $str;
}
```

28. Ábra: az űrlap érkező támadások semlegesítése

Végpontok

A szerveren csak **GET** és **POST** metódusú végpontok találhatóak. Ha az adott végpontba olyan metódusú kérés érkezik, amit a végponton nem engedélyeztünk, akkor a szerver visszaküld egy 405-ös „Method Not Allowed” hiba kódot.

Az alábbi táblázatokról az olvasható le, hogy az adott végpontok mely feltételek teljesülése mellett, milyen válasz küldenek vissza.

Az alábbi **GET** metódusú végpontok léteznek:

- API/index.php:

Bejelentkezett felhasználó: nincs	Bejelentkezett felhasználó: mark22
200 OK	200 OK
`"user": null`	`"user": "mark22"`

- API/account/index.php:

Bejelentkezett felhasználó: nincs	Bejelentkezett felhasználó: mark22
401 Unauthorized	200 OK "firstname": "Mark", "lastname": "Nagy", "username": "mark22", "email": "nagydotmark@gmail.com", "battles": "0", "wins": "0", "points": "0"

- API/auth/login/index.php:

  A végpont kiolvassa a szerver üzenetet (**flash**), ha van, akkor elküldi, ha nincs, akkor null-t küld.

Bejelentkezett felhasználó: nincs,		Bejelentkezett felhasználó: mark22
Szerver üzenet: nincs	Szerver üzenet: alma	
200 OK "message": null	200 OK "message": "alma"	403 Forbidden

- API/auth/logout/index:

  A végpont tud szerverüzenetet (**flash**) készíteni.

Bejelentkezett felhasználó: nincs	Bejelentkezett felhasználó: mark22
403 Forbidden	200 OK (Szerver üzenet: "Successfully logged out.")

- API/signup/index.php:

Bejelentkezett felhasználó: nincs	Bejelentkezett felhasználó: mark22
200 OK	403 Forbidden

Az alábbi **POST** metódusú végpontok léteznek:

- API/account/changepw.php:

<table>
<tr><td rowspan="3">Bejelentkezett felhasználó: nincs</td><td colspan="4">Bejelentkezett felhasználó: mark22</td></tr>
<tr><td rowspan="2">Volt hibásan kitöltött mező</td><td colspan="3">Nem volt hibásan kitöltött mező</td></tr>
<tr><td rowspan="2">Helytelen jelszó</td><td colspan="2">Helyes jelszó</td></tr>
<tr><td>Módosítás nem sikerült</td><td>Módosítás sikerült</td></tr>
<tr><td>401 Unauthorized</td><td>400 Bad Request
<code>"success": false "errors": {…} "message": "Invalid data, please fix it."</code></td><td>400 Bad Request
<code>"success": false "error": {…} "message": "Wrong password."</code></td><td>400 Bad Request
<code>"success": false "error": {…} "message": "Failed to change password."</code></td><td>200 OK
<code>"success": true "message": "Password successfully changed."</code></td></tr>
</table>

- API/auth/login/index.php

<table>
<tr><td rowspan="3">Bejelentkezett felhasználó: mark22</td><td colspan="3">Bejelentkezett felhasználó: nincs</td></tr>
<tr><td rowspan="2">Volt hibásan kitöltött mező</td><td colspan="2">Nem volt hibásan kitöltött mező</td></tr>
<tr><td>Az elküldött adatok helytelenek</td><td>Az elküldött adatok helyesek</td></tr>
<tr><td>403 Forbidden</td><td>400 Bad Request
<code>"success": false "errors": {…} "message": "Failed to LogIn."</code></td><td>400 Bad Request
<code>"success": false "message": "Wrong username or password."</code></td><td>200 OK
<code>"success": true "message": "Sucsessfully LoggedIn."</code></td></tr>
</table>

- API/account/modify.php

<table>
<tr><td rowspan="3">Bejelentkezett felhasználó: nincs</td><td colspan="4">Bejelentkezett felhasználó: mark22</td></tr>
<tr><td rowspan="2">Volt hibásan kitöltött mező</td><td colspan="3">Nem volt hibásan kitöltött mező</td></tr>
<tr><td rowspan="2">Helytelen jelszó</td><td colspan="2">Helyes jelszó</td></tr>
<tr><td>Módosítás sikerült</td><td>Módosítás nem sikerült</td></tr>
<tr><td>401 Unauthorized</td><td>400 Bad Request
<code>"success": false "errors": {…} "message": "Invalid data, please fix it."</code></td><td>400 Bad Request
<code>"success": false "error": {…} "message": "Wrong password."</code></td><td>400 Bad Request
<code>"success": false "error": {…} "message": "Failed to modify profile.</code></td><td>200 OK
<code>"success": true "message": "Profile successfully modified."</code></td></tr>
</table>

- API/game/index.php

<table>
<tr><td>Bejelentkezett felhasználó: nincs</td><td colspan="2">Bejelentkezett felhasználó: mark22</td></tr>
<tr><td rowspan="2">401 Unauthorized</td><td>Az elküldött adatok helytelenek</td><td>Az elküldött adatok helyesek</td></tr>
<tr><td>400 Bad Request</td><td>200 OK</td></tr>
</table>

- API/scoreboard/index.php

<table>
<tr><td>Bejelentkezett felhasználó: nincs</td><td>Bejelentkezett felhasználó: mark22</td></tr>
<tr><td>200 OK
<code>"battles": [...] "wins": [...] "points": [...]</code></td><td>200 OK
<code>"username": "mark22" "battles": [...] "wins": [...] "points": [...]</code></td></tr>
</table>

- Api/signup/index.php

  A végpont tud szerverüzenetet (**flash**) készíteni.

Bejelentkezett felhasználó: mark22	Bejelentkezett felhasználó: nincs		
	Volt hibásan kitöltött mező	Nem volt hibásan kitöltött mező	
		Belső hiba lépet fel	Belső hiba nem lépet fel
403 Forbidden	400 Bad Request `"success": false "errors": {…} "message": "Invalid data, please fix it."`	400 Bad Request `"success": false "message": "Faild to Sign Up."`	200 OK `"success": true "message": "Sucsessfuly Signed Up."` (Szerver üzenet: "LogIn into your new account.")

Frontend

React.js[8]

A React.js egy JavaScript könyvtár a felhasználói felületek készítéséhez. A Facebook fejlesztő csapata készítette 2013-ban. Mára a legnépszerűbb webes frontend **framework** egyike.

A React.js a JSX bevezetésével szakított azzal a gyakorlattal, miszerint a sablonnak és a logikának fizikailag el kell különülniük. Rámutatott arra, hogy jobban járunk, ha különálló nézetek és kontrollerek helyett komponenseket készítünk, melyek többször is felhasználhatóak és egység tesztelhetőek.[9]

React.js-ben miden DOM objektumhoz tartozik egy úgynevezett virtuális DOM objektum, ami az eredeti DOM objektum leegyszerűsített másolata. A virtuális DOM ugyanazokkal a tulajdonságokkal rendelkezik, mint az igazi, leszámítva az, hogy képtelen megváltozgatni azt ami a képernyőn van. A virtuális DOM-ra azért van szükségünk, mert sokkal gyorsabban lehet vele dolgozni, mit az igazi DOM objektummal.

Amikor kirajzolunk egy JSX elemet, minden virtuális DOM frissítésre kerül. A virtuális DOM működése rendkívül gyors. A React összehasonlítja az új és az előző virtuális DOM-ot, ebből megtudja, hogy az eredeti DOM melyik részét kell frissítenie, így nincs felesleges üzemidő, ezért rengeteg költséget tud megtakarítani.[10]

A program a create-react-app csomag használatával készült el. Amely az `npx create-react-app my-app` parancs futtatásával képes elkészíteni egy teljes értékű (myApp nevű) React alkalmazást. Automatikusan letölti és konfigurálja a Webpack és Babel csomagokat, amik az ES6, a JSX és a React szintaxist a böngészők számára is futtatható JavaScript kóddá alakítják.[11]

Függvények

A **classname** függvény úgy működik, hogy paramétereknek megkapja azokat a sztringeket, amikből az adott html elem class-át szeretnénk készíteni, majd egy összefüggő sztringet alkot belőlük. Ez azért fontos, mert így a sztringekhez egyenként tudunk `bool && expr` típusú feltételeket adni, melyek úgy működnek, hogy ha hamis az adott logikai változó értéke, egy hamis értéket, egyébként a kifejezés értékét adják vissza. Így a végső sztringünkben csak azok a class elemek szerepelnek, amiket mi az adott állapotban a html elem class-ának akarunk adni.

```
export function classname() {
    let classStr = '';
    for(let i = 0; i < arguments.length; i++) {
        if(!!arguments[i]) {
            classStr += arguments[i].trim() + ' ';
        }
    }
    return classStr.trim();
}
```

*29. Ábra: a **classname** függvény*

A **fetchAjax** nevű függvény felelős az aszinkron kérések kezeléséért. Paraméterben kapja meg, azt url-t és a kérés metódusát (GET, POST, …), továbbá azt a **Json** állományt amit el kell küldenie. Meg kell még adni számára egy **callback** függvényt, melyet siker esetén meg tud hívni a szervertől kapott válasszal.

A játék folyamatosságát biztosító időzítőt a **mainTimerFunc** függvény szolgáltatja. Paraméterül várja másodpercben megadva, a visszaszámlálás intervallumát, annak a html elemnek az azonosítóját, ahova kiírja másodpercenként a visszamaradó időt, illetve ő is elvár egy **callback** függvényt, amit az idő lejártával meghív. Az idő nyilvántartására szolgáló változót egy függvény **closures**-ben hozzuk létre, ez annyi tesz, hogy ezt a változót kívülről nem lehet elérni, így elindulás után már nem lehet módosítani, csak az időzítő tudja csökkenteni az értékét.[12]

Komponensek

Az alkalmazásban a komponensek úgy kerültek megírásra, hogy többször felhasználhatóak legyenek akár egy oldalon belül többször használjuk, akár különböző oldalakon helyezzük el őket.

A **dataPanel** komponens a szervertől érkező információk vizuális megjelenítésére szolgál. A kapott adatokból egy két oszlopos táblázatot készít, baloldalon a mező nevekkel, jobboldalon a hozzájuk tartozó értékekkel.

Talán a legösszetettebb komponens a **form**. Ez a komponens felel az űrlapok elkészítéséért és működéséért. Önmagában egy üres űrlap, amin egy gomb található az elküldéshez. A szöveges mezőket és a checkbox-okat egy paraméterben kapott konstans leíró fájl alapján dinamikusan készíti el. Itt kihívást jelent, hogy ezek mindegyike egy új különálló komponens, így azok az értékeket, amiket a felhasználók beadnak ezekben a komponensekben helyezkednek el. Erre a problémára jelent megoldást a **2 way binding**. Ebben az esetben a generált komponenseknek paraméterként adunk egy függvényt, amelyet ők maguk meg tudnak hívni paraméterben az értékükkel, így a **form**-ba vissza jut az értékük. Ezt követően ezekkel az adatokkal, a küldés gombot megnyomva meghívja a **fetchAjax** függvényt. A **form** komponens az alkalmazás négy helyen használja, mindig különböző mezőkkel.

A játék mezőket a **gameTable** komponens készíti. Itt kap igazán fontos szerepet a **classname** függvény, ugyanis itt a tábla modellje alapján kerülnek kiosztásra az egyes mezőkhöz tartozó class-ok. Illetve itt található az a logika, amely a mezők kiválasztásánál a vizuális segítséget szolgáltatja.

A **horisontalPills** és **verticalPills** nagyon hasonlóak működésükben. Mindketten a paraméternek kapott adatok alapján generálnak egy menürendszert.

Az **inputCheckBox** és **inputTextField** komponensek többszörös felhasználásával készítjük el az űrlapokat. Az értékük megváltozását folyamatosan nyomon követik az **onChange** eseményre. A **2 way binding** működéséhez paraméterűl kapott függvényt **onBlur** eseményre hívják meg az éppen aktuális értékükkel.

A **navBar** komponens készíti el az oldal tetején található menüsort. Az ezen megjelenő linkeket a **navBarLinks** komponensen helyezi el, attól függően, hogy melyik oldalon állunk, illetve, hogy be vagyunk-e jelentkezve, vagy sem.

Routing

Az alkalmazáshoz összesen nyolc oldal készült. A megfelelő oldal megjelenítéséért és az útvonalak kezelését a **react-router-dom**[13] **npm** csomag felel. Ez annyit jelent, hogy az oldalak között anélkül járhatunk, hogy le kéne hozzá tölteni html fájlokat. Ha a felhasználó olyan útvonalat ad meg, ami nem létezik, akkor betöltődik a **pageNotFound** oldal, amin egy link segítségével elérjük a főoldalt.

```
render() {
    return (
        <Switch>
            <Route path={PageURL.index} exact component={IndexPage} />
            <Route path={PageURL.login} component={LoginPage} />
            <Route path={PageURL.signup} component={SignupPage} />
            <Route path={PageURL.logout} component={LogoutPage} />
            <Route path={PageURL.account} component={AccountPage} />
            <Route path={PageURL.game} component={GamePage} />
            <Route path={PageURL.scoreboard} component={ScoreBoardPage} />
            <Route component={PageNotFound} />
        </Switch>
    );
}
```

30.Ábra: ***rounting*** *az App.js-ben*

Css

Az oldalhoz tartozó stílus szabályokat az **App.css** fájl tartalmazza. Ezt a fájlt az **App.scss** fájlból generálja az oldal, melybe importáljuk a **frontend/src/styles** mappában található scss fájlok tartalmát.

A Sass-nek köszönhetően sokkal egyszerűbben tudunk css szabályokat írni. Lehetővé teszi, hogy készítsünk változókat, egymásba illesztett szabályokat és többször felhasználható kódrészleteket, mind ezt szabványos css szintakszist használva.[14]

Websocket

Valósidejűség[15]

Egy webes alkalmazás attól lesz valós idejű, hogy az elküldött adatot a fogadó az elküldés pillanatában megkapja anélkül, hogy a küldőt folyamatosan kérdezgetnie kellene. Ennek köszönhetően a felhasználók számára valós idejű élmény nyújt.

A websocket egy protokoll, ami két irányú kommunikációs csatornát biztosít. Ez azt jeleni, hogy a böngésző és a web szerver valós idejű kommunikációt tudnak fenntartani. Üzeneteket küldhetnek és fogadhatnak miközben a kapcsolatuk folyamatosan nyitott.

Socket.IO[16]

A Socket.IOegy JavaScript könyvtár, melynek segítségével egyszerűen készíthetünk valós idejű webalkalmazásokat, melynek segítségével tudjuk a websoketet használni. Két részre különül el. A szerverre, ami a szerveren (host gép) futtat a Node.js és a kliensre, amit a felhasználók böngészője futtat. Az **npm** csomagkezelővel telepíthető.

Működése

A websoket szerver kiszolgálja és nyilvántartja a klienseket, akik az oldal betöltésekor automatikusan kapcsolódnak. A mikor egy felhasználó jelentkezik a „match making” folyamatba, akkor szerver egy külön listára teszi. A listára tétel előtt ellenőrzi, hogy olyan kliens aki már csatlakozott, vagy éppen játszik ne kerüljön be. Később erről a listáról sorsolja össze a felhasználókat a játékhoz. A játékba bekerült klienseket szintén külön listán kezeli. A szerver folyamatosan aktív kapcsolatban áll a kliensekkel, így ha valamelyikükkel megszakadna a kapcsolat a játék, vagy a sorsolás közben akkor erről rögtön informálódik, így szükség esetén képes értesíteni az érintett klienseket.

Deploy

Domain és IP-cím konfigurálása

Fejlesztés során a frontendet és aszervereket **localhost** futtatjuk, mivel így sokkal gyorsabban tudunk haladni a kód írásával, mintha minden egyes apró változtatás után **build**-et kéne készítenünk és azt kiraknánk egy külső szerverre. Másrészről előre nem tudatjuk, hogy mi annak a gépnek az IP címe, vagy domin neve amelyre a szerver szolgáltatások ki fognak kerülni. Az előbbi okok miatt a forráskódokba beleírtuk, hogy a frontend a backend végpontjait és a websocket szervert **localhost** keresse. Az elérési címet a forráskódban összesen három helyen kell kicserélnünk:

- A backend-en a **private/headers.php** harmadik sorában kell lecserélni a frontend főoldalának a címére.
- A frontend kódjában a **src\constants\serverUrl.js** fájl első sorában. Ide a backed végpontok gyökérnek a címét kell írni.
- Szintén a frontend-ben kell át állítani az a címet, amivel az alkalmazás eléri a websocket-et. Ezt a **src\actions\wsclient.js** harmadik sorában tehetjük meg.

Az alábbi ábrán az látható, hogy milyen módon kell kicserélni a címeket.

```
export const socket = openSocket('http://localhost:8080');
```

*31.a Ábra: a kliens a szervert a **localhost** 8080-as porton keresi*

```
export const socket = openSocket('http://192.168.0.7:8080');
```

*31.b Ábra: a címet a **localhost**-ról módosítottuk egy másik gép IP címére*

Build készítése

A beckend és a webszoket szervereket a címek beállítása után, bármi nemű módosítás nélkül kitehetjük a kiszolgáló gépre. Viszont a frontend-del nem ez a helyzet, ebből **build**-et kell készítenünk.

Ezt a **build**-et úgy készíthetjük el, ha a frontend gyökér könyvtárában futtatjuk **az npm run-script build** parancsot. Ekkor a könyvtárszerkezetünkben megjelenik egy „build” nevű mappa, ez tartalmazza a ki **build**-elt fájlokat.

Tesztelés

Backend

A szerver végpontjainak a tesztelése manuálisan történt.

- **API/**

Teszt eset	Teszt kimenete
Bejelentkezett felhasználó: nincs, metódus: **GET**	200 OK `{"user": null}`
Bejelentkezett felhasználó: mark22 metódus: **GET**	200 OK `{"user": "mark22"}`
Bejelentkezett felhasználó: mark22 metódus: **POST**	405 Method Not Allowed

- **API/account/**

Teszt eset	Teszt kimenete
Bejelentkezett felhasználó: nincs, metódus: **GET**	401 Unauthorized

Bejelentkezett felhasználó: mark22 metódus: **GET**	200 OK
	`{"firstname": "Mark",`
	`"lastname": "Nagy",`
	`"username": "mark22",`
	`"email":`
	`"nagydotmark@gmail.com",`
	`"battles": "0",`
	`"wins": "0",`
	`"points": "0"}`

Bejelentkezett felhasználó: mark22 metódus: **POST**	405thod Not Allowed

- **API/account/changepw.php**

Teszt eset	Teszt kimenete
Bejelentkezett felhasználó: nincs, metódus: **GET**	405 Method Not Allowed
Bejelentkezett felhasználó: nincs metódus: **POST** {"oldpassword":"123456", "newpassword":"123456", "newpassword2":"123456"}	401 Unauthorized
Bejelentkezett felhasználó: mark22 metódus: **POST** {"oldpassword":"", "newpassword":"", "newpassword2":""}	400 Bad Request {"success": false, "error":{ "newpassword": "6 character long password required.", "oldpassword": " Password reqired for confirm. " }, "message": "Wrong password."}
Bejelentkezett felhasználó: mark22 metódus: **POST** {"oldpassword":"badpassword", "newpassword":"123456", "newpassword2":"123456"}	400 Bad Request {"success": false, "error":{ "oldpassword": "Wrong password." }, "message": "Wrong password."}

Bejelentkezett felhasználó: mark22 metódus: **POST** {"oldpassword":"123456", "newpassword":"654321", "newpassword2":""}	400 Bad Request {"success": false, "error":{ "newpassword2": "New password again required." }, "message": "Invalid data, please fix it."}
Bejelentkezett felhasználó: mark22 metódus: **POST** {"oldpassword":"123456", "newpassword":"654321", "newpassword2":"123456"}	400 Bad Request {"success": false, "error":{ "newpassword2": "Passwords don't match." }, "message": "Invalid data, please fix it."}
Bejelentkezett felhasználó: mark22 metódus: **POST** {"oldpassword":"123456", "newpassword":"654321", "newpassword2":"654321"}	200 OK {"success": true, "message": "Password successfully changed."}

- **API/account/modify.php**

Teszt eset	Teszt kimenete
Bejelentkezett felhasználó: mark22, metódus: **GET**	405 Method Not Allowed
Bejelentkezett felhasználó: nincs metódus: **POST** {"firstname":"Pista", "lastname":"Kiss", "email":"kiss@pista.hu", "password":"123456"}	401 Unauthorized
Bejelentkezett felhasználó: mark22 metódus: **POST** {"firstname":"Pista", "lastname":"Kiss", "email":"kiss@pista.hu", "password":""}	400 Bad Request "success": false, "errors":{ "password": "Password required." }, "message": "Invalid data, please fix it."
Bejelentkezett felhasználó: mark22 metódus: **POST** {"firstname":"Pista", "lastname":"Kiss", "email":"kiss@pista.hu", "password":"not"}	400 Bad Request "success": false, "error": { "password": "Wrong password." }, "message": "Wrong password."

Bejelentkezett felhasználó: mark22 metódus: **POST** {"firstname":"Pista!", "lastname":"Kiss23", "email":"kiss@pista ", "password":"123456"}	400 Bad Request "success": false, "errors": { "firstname": "Only letters allowed.", "lastname": "Only letters allowed.", "email": "Email looks invalid. Use an other one." }, "message": "Invalid data, please fix it."
Bejelentkezett felhasználó: mark22 metódus: **POST** {"firstname":"Pista", "lastname":"", "email":"", "password":"123456"}	200 OK "success": true, "message": "Profile successfully modified."
Bejelentkezett felhasználó: mark22 metódus: **POST** {"firstname":"", "lastname":"Kiss", "email":"kiss@pista.hu", "password":"123456"}	200 OK "success": true, "message": "Profile successfully modified."

- **API/auth/login**

Teszt eset	Teszt kimenete
Bejelentkezett felhasználó: nincs, metódus: **GET**	200 OK "message": null

Bejelentkezett felhasználó: nincs, előzőleg jelentkeztünk ki metódus: **GET**	200 OK "message": "Successfully logged out."
Bejelentkezett felhasználó: nincs, metódus: **POST** {"username":"", "password":""}	400 Bad Request "success": false, "errors": { "username": "Username required.", "password": "Password required." }, "message": "Failed to LogIn."
Bejelentkezett felhasználó: nincs, metódus: **POST** {"username":"notuser", "password":"notpassword"}	400 Bad Request "success": false, "message": "Wrong username or password."
Bejelentkezett felhasználó: nincs, metódus: **POST** {"username":"mark22", "password":"notpassword"}	400 Bad Request "success": false, "message": "Wrong username or password."
Bejelentkezett felhasználó: nincs, metódus: **POST** {"username":"mark22", "password":"123456"}	200 OK "success": true, "message": "Sucsessfully LoggedIn."
Bejelentkezett felhasználó: mark22, metódus: **GET**	403 Forbidden
Bejelentkezett felhasználó: mark22, metódus: **POST** {"username":"mark22", "password":"123456"}	403 Forbidden

Bejelentkezett felhasználó: mark22, metódus: **DELETE**	405 Method Not Allowed

- **API/auth/logout/**

Teszt eset	Teszt kimenete
Bejelentkezett felhasználó: nincs, metódus: **GET**	403 Forbidden
Bejelentkezett felhasználó: mark22, metódus: **GET**	200 OK
Bejelentkezett felhasználó: mark22, metódus: **PUT**	405 Method Not Allowed

- **API/game/**

Teszt eset	Teszt kimenete
Bejelentkezett felhasználó: mark22, metódus: **GET**	405 Method Not Allowed
Bejelentkezett felhasználó: nincs, metódus: **POST** `{"isWin":""}`	401 Unauthorized
Bejelentkezett felhasználó: mark22, metódus: **POST** `{"isWin":""}`	400 Bad Request
Bejelentkezett felhasználó: mark22, metódus: **POST** `{"isWin":"true"}`	400 Bad Request
Bejelentkezett felhasználó: mark22, metódus: **POST** `{"isWin":true}`	200 OK
Bejelentkezett felhasználó: mark22, metódus: **POST** `{"isWin":false}`	200 OK

- **API/scoreboard/**

Teszt eset	Teszt kimenete
Bejelentkezett felhasználó: mark22, metódus: **GET**	405 Method Not Allowed

Bejelentkezett felhasználó: mark22, metódus: **POST**	200 OK
	`"username": "mark22"`
	`"battles": [...]`
	`"wins": [...]`
	`"points": [...]`

Bejelentkezett felhasználó: nincs, metódus: **POST**	200 OK
	`"battles": [...]`
	`"wins": [...]`
	`"points": [...]`

Bejelentkezett felhasználó: nincs, metódus: **DELETE**	405 Method Not Allowed

Frontend[17]

A frontend tesztelése automatizált teszt fájlokkal biztosított, ezeket a fájlokat a frontend gyökér könyvtárában állva az **npm test** paranccsal tudjuk futtatni parancssorban, vagy PowerShell ablakban.

A teszteknek a mellett, hogy garantálják az adott komponens, vagy függvény hiba mentes működését, fontos szerepet játszanak abban, hogy segítsék a fejlesztőnek megérteni a tesztelt kódrészlet pontos működését. Ehhez persze jól olvasható és karbantartott teszt bázisra van szükségünk. Továbbá fontos, hogy olyan teszteket hozzunk létre, amelyek azt vizsgálják, hogy a különböző komponensek, vagy függvények adott bemenetre az elvárt kimenetet adják, és ne azt, hogy milyen html elemek követik egymást, mivel ezek könnyen eltörhetnek.

A teszt jól olvashatóságát szemelőt tartva a teszt elején tároljuk el a bemeneti változókat egy konstansba, így a teszt elején máris látható, hogy az adott kódrészletnek milyen bemenetei lehetnek. Ezeket ekkor érdemes üresre, vagy nullára állítani, hogy később ne zavarjanak be.

```
describe('<form />', () => {
    const props = {
        title: null,
        submitText: null,
        fields: null,
        checkBoxs: null,
        redirectURL: null,
        triggerRefresh: () => {},
    };
```

32. Ábra: a teszt elején komponens összes paraméterét nullára vagy hamisra állítjuk

Ezután minden egyes teszt esetben vegyünk fel egy olyan külön (lokális) konstanst vagy változót, amelyben csak azok a változó kapnak értéket, amelyekkel az adott teszt eset dolgozik. Ezáltal növekszik az átláthatóság, és teszt törés esetén könnyen látható, hogy melyik bemenet okozza a hibát.

```
it('renders emtpy form right', () => {
    const localProps = {
        ...props,
        title: 'Form',
        submitText: "Submit",
    };
    const wrapper = shallow(<Form {...localProps} />);
    expect(wrapper.find('h3').text()).to.equal('Form');
    expect(wrapper.find('InputTextField')).to.not.exist;
    expect(wrapper.find('InputCheckBox')).to.not.exist;
    expect(wrapper.find('input[type="submit"][value="Submit"]')).to.exist;
    expect(wrapper.find('.alert')).to.not.exist;
    expect(wrapper.find('Redirect')).to.not.exist;
});
```

33. Ábra: csak a tesztelt paramétereknek adunk értéket

Összegzés

Megvalósított célok

Véleményem szerint az eredetileg kitűzött célokat elérte a program. A weboldalon ki-, és be jelentkezhetnek, regisztrálhatnak a felhasználók. Bejelentkezve lehetőségük van játszani. Az ellenfél kisorsolása és a teljes játék valós időben történik. A végeredménynek megfelelően a program statisztikát vezet a játékosokról.

Az oldal reszponzívitásának köszönhetően a kisebb felbontású képernyőt, vagy okostelefont használók számára is használható az oldal.

Továbbfejlesztési lehetőségek

Az oldalhoz adhatnánk **tracker**-eket, melyek segítségével privát statisztikákat gyűjthetnénk arról, hogy a felhasználóink milyen módon használják az oldal funkcióit. Ezen információk alapján átfogó képet kaphatunk az egyes funkciók kihasználtságáról.

Jelenleg a program csak a két-játékos módot támogatja. Készíthetnénk egy olyan egy játék módot, ahol a játékos egy **deep learning**-en alapuló algoritmus ellen játszik. A programot folyamatosan taníthatnánk a két-játékos mód figyelésével.

Hivatkozások

[1] PHP
[Hivatkozva: 2018.05.12]
http://php.net/

PHP, Wikipédia a szabad enciklopédia
[Hivatkozva: 2018.05.12]
https://hu.wikipedia.org/wiki/PHP

PHP alapok, Web programozás
[Hivatkozva: 2018.05.12]
http://web.progtanulo.hu/web-programozas-alapismeretek/3-szerver-oldali-mukodes/32-php-alapok

[2] PHP MySQL Database, W3scool.com
[Hivatkozva: 2018.05.12]
https://www.w3schools.com/php/php_mysql_intro.asp

[3] Node.js
[Hivatkozva: 2018.05.12]
https://nodejs.org/en/

npm
[Hivatkozva: 2018.05.12]
https://www.npmjs.com/

[4] What is npm?, npm Documentation
[Hivatkozva: 2018.05.12]
https://docs.npmjs.com/getting-started/what-is-npm

[5] XAMPP
[Hivatkozva: 2018.05.14]
https://www.apachefriends.org/hu/index.html

[6] Márton Visnovitz: GitHub
[Hivatkozva: 2018.05.14]
https://github.com/vimtaai/elte/blob/master/2016-17-2/wf2-7/gyak11/functions.php

[7] React.js
[Hivatkozva: 2018.05.12]
https://reactjs.org/

[8] PHP 5 Form Validation, W3scool.com
[Hivatkozva: 2018.05.14]
https://www.w3schools.com/php/php_form_validation.asp

[9] Jens Neuhaus: Angular vs. React vs. Vue: A 2017 comparison, Medium
[Hivatkozva: 2018.05.12]
https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176

[10] React: The Virtual DOM, Codecademy
[Hivatkozva: 2018.05.12]
https://www.codecademy.com/articles/react-virtual-dom

[11] Create React App, GitHub
[Hivatkozva: 2018.05.12]
https://github.com/facebook/create-react-app

[12] JavaScript Closures, W3scool.com
[Hivatkozva: 2018.05.12]
https://www.w3schools.com/js/js_function_closures.asp

[13] react-router-dom, npm
[Hivatkozva: 2018.05.12]
https://www.npmjs.com/package/react-router-dom

[14] Sass
[Hivatkozva: 2018.05.12]
https://sass-lang.com/

[15] Luis Aviles: Real Time Apps with TypeScript: Integrating Web Sockets, Node & Angular, Daily JS
[Hivatkozva: 2018.05.13]
https://medium.com/dailyjs/real-time-apps-with-typescript-integrating-web-sockets-node-angular-e2b57cbd1ec1

[16] Socket.IO
[Hivatkozva: 2018.05.12]
https://socket.io/

[17] Benedek Gagyi: Check out these beginner-friendly unit testing patterns for React, freecodecamp
[Hivatkozva: 2018.05.12]
https://medium.freecodecamp.org/unit-testing-patterns-for-react-720a8275873b