ETDK-dolgozat

Csibi Zsolt

Jánosi Borbála

Nagy Tifani-Franciska

XXV. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)

Kolozsvár, 2022. május 19–22.

Athletimeter

Atlétika versenyek szervezését támogató szoftverrendszer

Szerzők:

Csibi Zsolt

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Jánosi Borbála

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Nagy Tifani-Franciska

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:

dr. Simon Károly, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Hegedüs Hunor, szoftverfejlesztő,

Codespring

Szász Erika, szoftverfejlesztő,

Codespring

Kivonat

Az Athletimeter projektet a székelyudvarhelyi Török Béla Atlétika Emlékverseny inspirálta. A verseny lebonyolításához szükséges adatok kezelése eddig "papíron" történt, ami megnehezítette a szervezők és a nézők számára az eredmények követését. A projekt célja, hogy ezt a folyamatot digitalizálja, ezáltal megkönnyítve az adatok feldolgozását.

A szervezőknek lehetőségük van egy webes és mobiltelefonos alkalmazáson keresztül versenyszámok, versenyzők és eredmények felvezetésére és kezelésére. A mobilalkalmazás egy beépített stopperórát és egy QR kód olvasót is biztosít, ezzel megkönnyítve az eredmények rögzítését, illetve a versenyzők azonosítását.

A verseny nézőinek lehetőségük van valós időben látni a különböző versenyszámok jelenlegi állását a webes alkalmazás megfelelő oldalán, vagy a verseny helyszínén kivetítve.

A szoftverrendszer három részre osztható: egy mobil alkalmazásra, egy webalkalmazásra, illetve egy felhőben futó szerverre.

A dolgozat bemutatja az alkalmazás működését, részletezve a funkcionalitásait, architektúráját és különböző komponenseit, valamint ismerteti a felhasznált technológiákat és eszközöket.

Tartalomjegyzék

Bevezető **1**

1. Az Athletimeter alkalmazás **3**

1.1. Szerepkörök és funkcionalitások 3

1.1.1. Vendégfelhasználó 3

1.1.2. Szervező 4

1.1.3. Adminisztrátor 6

1.2. Kommunikációs modell 7

2. Az alkalmazás felépítése **9**

2.1. Alkalmazásszerver 9

2.1.1. Szerver struktúrája 9

2.1.2. Szerver architektúrája 11

2.1.3. Szerverhez érkező kérések feldolgozása 12

2.2. Web kliens 12

2.2.1. A kliens struktúrája 12

2.2.2. Architektúra 14

2.2.3. Komponensek 15

2.2.4. Horgok 15

2.2.5. Élő frissítés 18

2.2.6. Biztonság 19

2.3. Mobil kliens 20

2.3.1. A mobil kliens struktúrája 20

2.3.2. Kommunikáció a szerverrel 21

2.3.3. Navigáció 22

2.3.4. Biztonság 22

3. Technológiák és eszközök **23**

3.1. Azure felhőszolgáltatás 23

3.1.1. Összehasonlítás más felhőszolgáltatókkal 23

3.2. Szerver technológiák 24

3.3. Web technológiák 25

3.4. Mobil technológiák 27
3.5. Eszközök 28

4. Az alkalmazás működése **29**
4.1. Szerver beüzemelése 29
4.2. Web kliens működése 29
4.3. Mobil kliens működése 33

Következtetések és továbbfejlesztési lehetőségek **37**

Bevezető

Az atlétika futó, ugró, dobó, gyalogló vagy lökő sportágakat magába foglaló sportesemény csoport. A motorikus képességek és kapcsolódó készségek fejlesztését célzó sportág-csoport előnye, hogy versenyei egyszerűen megszervezhetőek, nem szükséges drága felszerelések beszerzése. [15] A dobó vagy ugró sportágak eredményeit adott számú próbálkozásból a legnagyobb távolság képezi, míg futás vagy gyaloglás esetén a mért idő, vagy a célba érés sorrendje határozza meg a helyezést. Az atlétikaversenyek, mozgásanyaguknak és egyszerűségüknek köszönhetően, nagy népszerűségnek örvendenek, ezért számos ilyen jellegű eseményt szerveznek meg minden évben, amelyeken gyerekek és felnőttek egyaránt részt vesznek, hogy kipróbálják magukat a különböző versenyszámokban.

A Török Béla Atlétika Emlékverseny Székelyudvarhelyen kerül évente megrendezésre, keretében különböző helységekből érkező gyerekek mérkőznek meg egymással. A versenyzők korosztályuknak megfelelően vesznek részt a különböző versenyszámokban. A sportoló futási idejét vagy az ugráson, dobáson elért távolságát a hozzá rendelt szervező méri, majd egy papírra jegyzi fel ezt. Ezután a résztvevő feladata elmenni a megfelelő pulthoz, és megmutatni az eredmények táblázatba foglalásáért felelős szervezőnek a papírra írt értéket. Miután minden esemény lezajlott, a kitöltött táblázatokat szétosztják a gyerekekért felelős képviselőknek. Ez a módszer eddig sok szervezőt igényelt, megnehezítette számukra az adatok kezelését és archiválását, valamint a nézők számára az eredmények követését, ráadásul a szervezés során előfordult emberi hibák száma is jelentős volt.

Az Athletimeter szoftverrendszert a Török Béla Atlétika Emlékverseny inspirálta, és célja a szervezők munkájának megkönnyítése, amelyet az adatok digitális feldolgozásával valósít meg. Az alkalmazás három alkotóelemre bontható: egy webes felület az adatok megtekintésére és kezelésére, egy mobiltelefonos applikáció az eredmények gyors felvezetésére, valamint egy felhő alapú szerver, amely a verseny időtartama alatt lesz elérhető, és feladata az említett platformok adatokkal való kiszolgálása.

A szervezők a webes alkalmazáson keresztül könnyen meg tudják valósítani versenyszámok, versenyzők és eredmények karbantartását. Az adatok hosszú távú megőrzésének céljából a felület különböző formátumba való mentést biztosít, amelyeket később a szervezők továbbíthatnak vagy különböző statisztikákat készíthetnek belőlük. Mivel a rendszer szervere a verseny előtt kerül beüzemelésre, a szervezők megfelelő formátumú adathalmazok feltöltésével képesek olyan, esetleg korábbi atlétikaversenyekről származó

adatok gyors hozzáadására, amelyek különböző versenyek esetén is ugyanazok lehetnek, mint például a korosztályok vagy a versenyszámok. A webalkalmazás segítségével a verseny nézőinek lehetőségük van az eredmények valós idejű követésére, amely akár a helyszínen is kivetítésre kerülhet.

Az eredmények hatékony felvezetéséhez a szoftverrendszer mobilos alkalmazása járul hozzá. Az alkalmazás egy QR[1] kód olvasóval van ellátva, ezáltal a szervezők a ruhára felragasztott kód beolvasásával azonosítani tudják a versenyzőt. Megfelelő felületen lehetőség van az eredmény megadására, illetve időben mérendő versenyszámok esetén egy beépített stopperórát lehet használni, amelynek leállítását, és a sportoló beazonosítását követően automatikusan menti a rendszer az eredményt.

A dolgozat hátralevő tartalma 5 fő fejezetre tagolható: az 1. fejezet bemutatja a szoftverrendszeren belül meghatározott felhasználói szerepköröket és az ezekhez tartozó funkcionalitásokat, valamint áttekintést nyújt a projekt kommunikációs modelljéről. A 2. fejezet részletesen tárgyalja az alkalmazás architektúrájának különböző komponenseit. A 3. fejezet ismerteti a felhasznált technológiákat és eszközöket. A 4. fejezet az alkalmazás beüzemelését és működését szemlélteti, az utolsó fejezetben pedig következtetések és néhány továbbfejlesztési lehetőség kerül megfogalmazásra.

A projekt fejlesztése 2021 júliusában, a U-hub Mentorprogram[2] keretén belül, a Codespring cégnél végzett nyári szakmai gyakorlat ideje alatt kezdődött. A 2021-2022-es egyetemi tanév első félévében, a csoportos projekt keretén belül a fejlesztés tovább folytatódott. Köszönet illeti mindazokat, akik segítségünkre voltak: témavezető tanárunkat, Dr. Simon Károly egyetemi adjunktust a szakmai irányításért, Hegedüs Hunor és Szász Erika szoftverfejlesztőket a mentorálásért, valamint Bartus Levente szoftverfejlesztőt a Codespring részéről a projekt ötletéért. Továbbá köszönet Erdei Csongornak és Parajdi Zoárdnak, akik a csoportos projekt alatt járultak hozzá az alkalmazás megvalósításához.

[1] Quick Response - Kétdimenziós vonalkód, amely információtárolásra használt.
[2] U-hub Mentorprogram oldala: https://u-hub.ro/mentorprogram/

1. Az Athletimeter alkalmazás

Az Athletimeter szoftverrendszer egy atlétikaverseny adatainak digitális kezelését a weboldalon és a mobilalkalmazáson keresztül biztosítja, amelyek egy felhőben futó szerverrel állnak kapcsolatban.

A rendszer funkcionalitásai, valamint komponenseinek kommunikációja az első fejezetben kerül részletes bemutatásra.

1.1. Szerepkörök és funkcionalitások

A rendszerben három különböző szerepkör határozható meg: vendégfelhasználó, szervező és adminisztrátor.

A rendszer webes felületén keresztül azoknak az adatoknak a karbantartása valósítható meg, amelyekre egy atlétikaverseny épül. Az adminisztrátor a platformon keresztül szervezőket tud hozzáadni a rendszerhez. Ezek számára lehetőség van korosztályok, résztvevők, versenyszámok és az ezekhez tartozó eredmények felvezetésére és kezelésére. Továbbá a weboldal a szurkolói élmény fokozását is biztosítja az eredmények valós idejű követésének lehetőségével. Ezeket a szolgáltatásokat a három szerepkör szerint lehet csoportosítani.

A mobilalkalmazás az eredmények hozzáadására nyújt hatékony alternatívát. Ebben az esetben a funkcionalitások kizárólag regisztrált felhasználók számára érhetőek el, akik a szervezők szerepét töltik be.

A felsorolt szerepkörök között különbség áll fenn jogosultságok terén. A vendégfelhasználó rendelkezik a legkevesebb joggal, ezt követi a hierarchiában a szervező, a legtöbb funkcionalitást pedig az adminisztrátor képes használni.

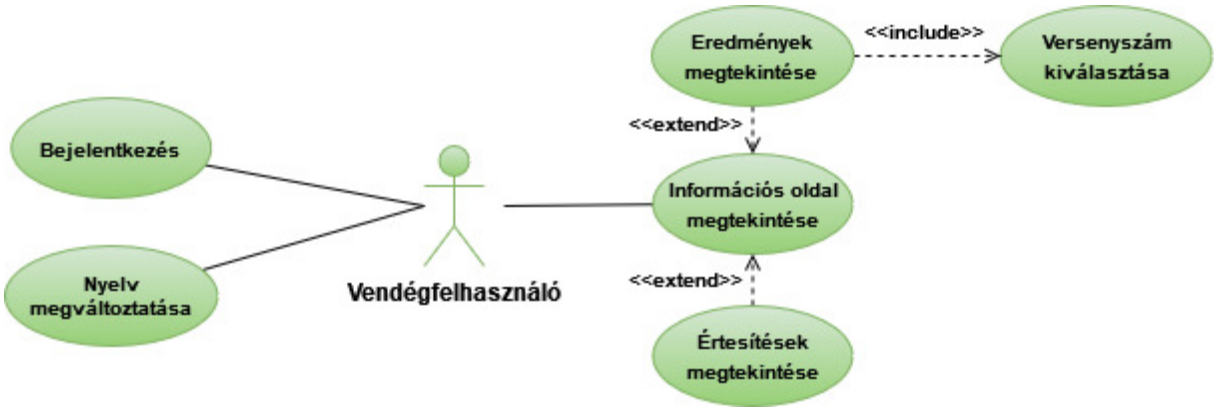
1.1.1. Vendégfelhasználó

A vendégfelhasználók nem tudják használni a mobilalkalmazás szolgáltatásait, azonban a weboldal biztosít néhány funkcionalitást számukra. Ezeket a 11. ábra szemlélteti.

Bejelentkezés vagy regisztráció nélkül a felhasználók vendégnek tekinthetőek, és kizárólag nézőként tudják használni a weboldalt.

Lehetőségük van az oldal nyelvének megváltoztatására, amely jelenleg a magyar és angol nyelvet támogatja.

Megtekinthetik az információs oldalt, amelynek segítségével a versenyállások kialakulása



1. ábra. Vendégfelhasználók számára elérhető funkcionalitások a webalkalmazásban.

figyelhető meg. Kiválaszthatják azt a versenyszámot, amelyben érdekeltek, majd ezt követően egy lista formájában láthatják a már megszületett eredményeket. Ha a megtekintés ideje alatt egy új eredmény került be a rendszerbe, a lista automatikusan frissül. Bizonyos esetekben az új eredményekről értesítéseket is kaphatnak, és ezáltal könnyebben nyomon tudják követni az állásokat.

1.1.2. Szervező

Egy atlétikaverseny folyamán a szervezők feladata az adatok megfelelő felvezetése a rendszerbe és karbantartása, valamint az eredmények gyors és helyes elmentése. Ezek teljesítésére, a webes felületen, a vendégfelhasználók jogai mellett, új szolgáltatásokat is elérnek, emellett hozzáférésük van a mobil alkalmazáshoz is.

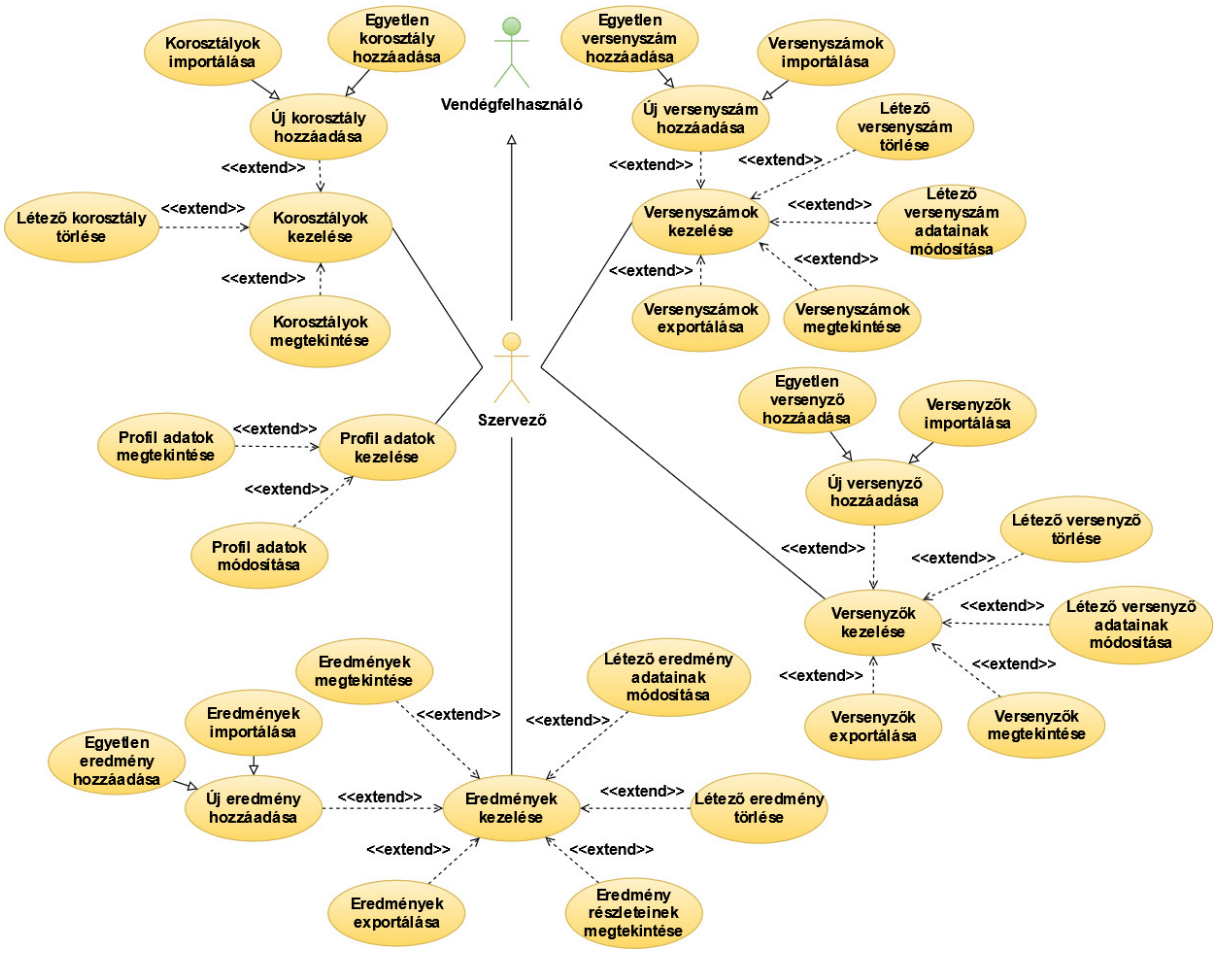
A weboldalra szervezőként bejelentkezett felhasználók számára a 2. ábrán látható funkcionalitások vannak biztosítva.

Kezelni tudják a korosztályokat, listázhatják vagy törölhetik azokat, vagy létrehozhatnak újakat, megadva egy egyedi nevet és egy kor intervallumot, amelyre az adott korosztály érvényes. Egy helyes JSON[3] állomány megadásával több korosztályt egyszerre hozzá tudnak adni a rendszerhez.

Emellett résztvevőket, versenyszámokat és eredményeket is karbantarthatnak. Listázhatják ezeket, és szűrni tudnak különböző kritériumok szerint, valamint PDF[4] és JSON formátumba való exportálásra is lehetőségük van. Létrehozhatnak új versenyzőket és versenyszámokat, vagy módosíthatnak létezőket. Egyező nem és korosztály esetén egy adott résztvevőhöz és

[3]JavaScript Object Notation - Szabványos szövegalapú formátum, amelyet adattárolásra és adatszállításra használnak.

[4]Portable Document Format - Fájlformátum, amely dokumentumok tárolására alkalmas.



2. ábra. Szervezők számára elérhető funkcionalitások a webalkalmazásban.

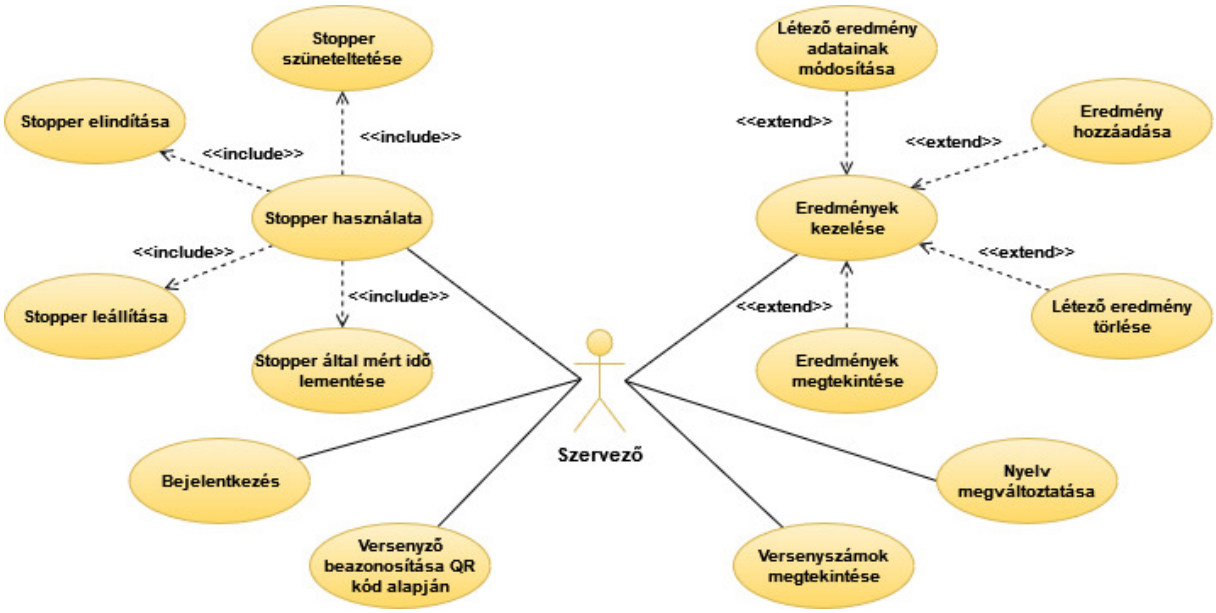
versenyszámhoz eredményt vezethetnek fel. Az esemény mértékegységének függvényében, az eredmény egy méterben vagy időben kifejezett érték, amelyet később szükség esetén felülírhatnak. Hasonlóképpen, a versenyzőkre, eseményekre és eredményekre JSON formátumból való importálást is végrehajthatnak. Emellett inaktívvá tehetik őket, azonban versenyek és eredmények esetén ez csak akkor megengedett, ha a műveletet végrehajtani kívánó szervező hozta létre előzőleg az adott versenyszámot.

A profiloldalon a felhasználójukhoz tartozó információkat kezelhetik. Megtekinthetik a felhasználónevüket, szerepkörüket és az általuk felvezetett versenyszámokat. Mindemellett módosíthatják a teljes nevüket és jelszót is cserélhetnek.

A mobilalkalmazás a 3. ábrán szemléltetett funkcionalitásokat nyújtja a szervezők számára.

A platform, a webes felülethez hasonlóan, támogatja a többnyelvűséget, így a szervezők bármikor módosíthatják a nyelvet (jelenleg a magyar és angol nyelvek támogatottak).

A megfelelő oldalon listázhatják a versenyszámokat, és különböző opciók kiválasztásával tudják szűkíteni a találatokat. Megtekinthetik egy versenyszám részleteit egy külön ablakban,



3. ábra. Szervezők számára elérhető funkcionalitások a mobilalkalmazásban.

amelyen az eseményhez tartozó eredmények is megjelennek.

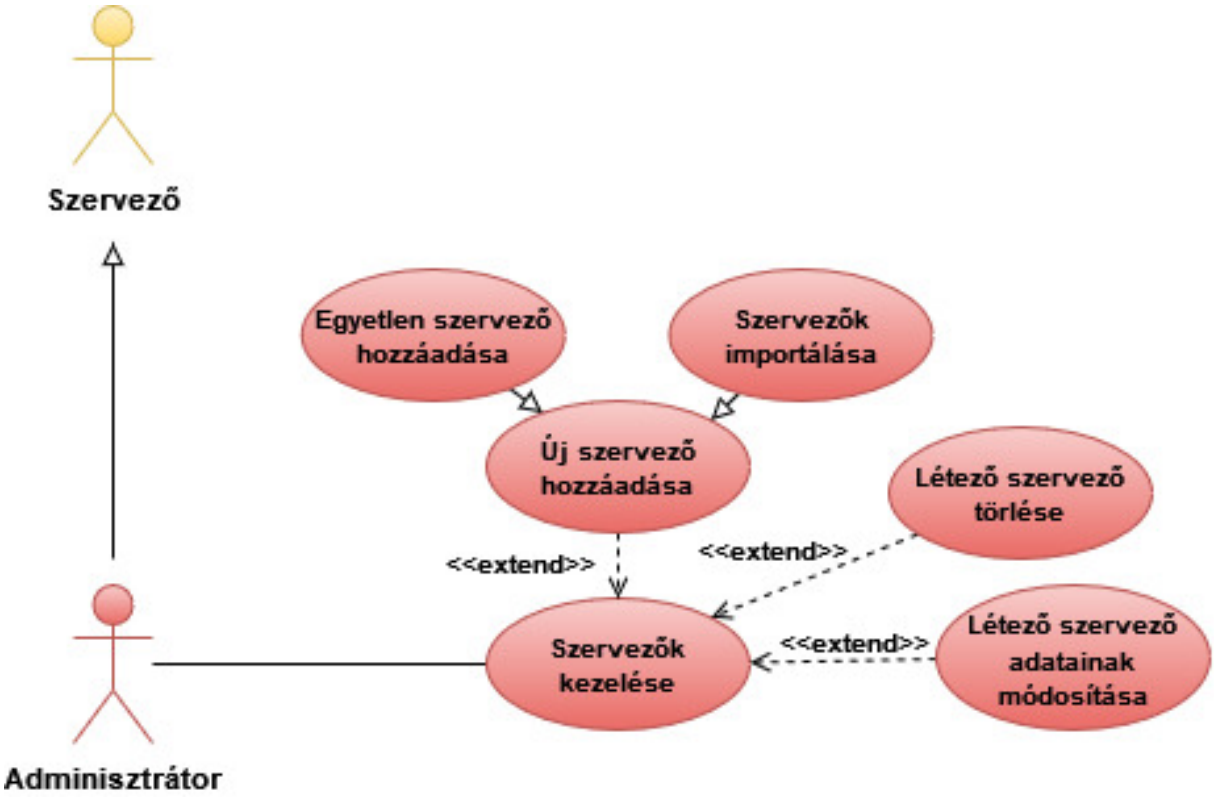
Az eredmények felvezetését többféleképpen is megvalósíthatják. Időben mérendő események esetén az applikációba beépített stopperórát használhatják, amelynek leállítása után a mért idő egy űrlapot tartalmazó oldalra lesz továbbítva. Ugyanakkor, az űrlap kézi kitöltésével is megadhatják az eredményt, a versenyszám típusától függetlenül. Az eredmények felvezetését segítő oldalon egy lenyíló listából kiválaszthatják a versenyszámot és a versenyzőt, akihez az eredmény tartozni fog, illetve az alkalmazás QR kód olvasóját is használhatják a sportoló beazonosításának érdekében. A rendszerbe bevezetett eredményeket listázhatják és szűrhetik, törölhetik vagy módosíthatják.

1.1.3. Adminisztrátor

A webalkalmazás esetében az adminisztrátor rendelkezik a szervezők jogaival, emellett rendelkezésére állnak új szolgáltatások is. Ezeket a 4. ábra szemlélteti.

Feladata egy atlétikaverseny szervezőinek kezelése. Megtekintheti az összes szervezőt, hozzáadhat a rendszerhez újakat a megfelelő űrlap kitöltésével, vagy egy érvényes JSON formátumú állomány megadásával. Inaktívvá tudja tenni őket, valamint módosíthatja a nevüket és a felhasználónevüket.

A mobilalkalmazáson belül nincs külön adminisztrátor szerepkör definiálva. Hozzáférhet az applikációhoz és a szervezők számára elérhető funkcionalitásokat használhatja.



4. ábra. Adminisztrátor számára elérhető funkcionalitások a webalkalmazásban.

1.2. Kommunikációs modell

A rendszer *kliens-szerver* kommunikációs modellre épül. A kliens feladata kéréseket küldeni a szervernek, és a szervertől kapott információkat megjeleníteni egy felhasználói felületen, míg a szerver a kapott kérések feldolgozásáért és a válasz küldéséért felelős.

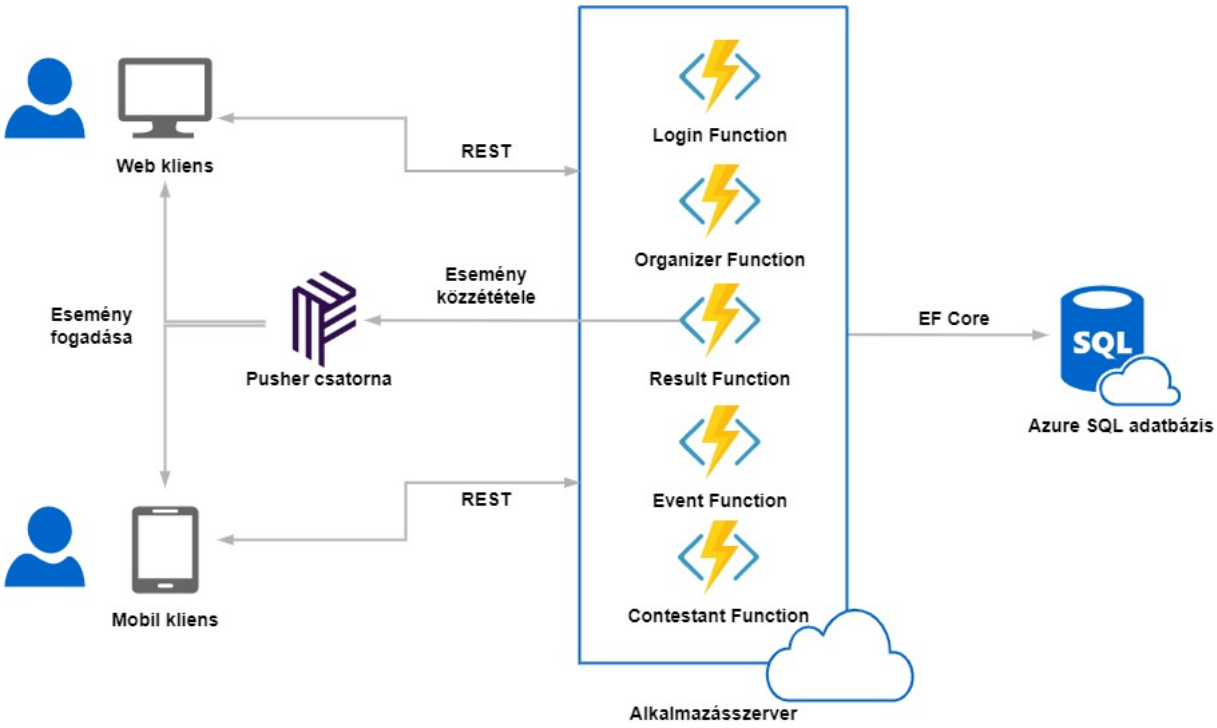
A felhasználók, szerepkörüknek megfelelően, a kliens platformokkal állnak közvetlen kapcsolatban. Ezek egy React keretrendszerre épülő webalkalmazás, illetve egy React Native-ban megvalósított mobil applikáció.

A rendszer szerverét öt darab .NET Core-ban implementált Azure függvényalkalmazás képezi. Ezek az Entity Framework Core által nyújtott szolgáltatások segítségével állnak kapcsolatban az adatokat tároló felhőalapú SQL Server adatbázissal. A rendszerben meghatározott adattípusok kezelésével (versenyzők, versenyszámok, eredmények, szervezők), valamint a bejelentkezéssel egy-egy külön függvényalkalmazás foglalkozik, amelyek egymástól függetlenül működnek és különböző URL[5]-ken fogadják a kéréseket.

A szerver és a web- illetve mobilalkalmazás közötti adatcsere a HTTP[6] protokoll szabályai

[5]Uniform Resource Locator - Szabványosított webcím, amely interneten elérhető erőforrások helyére hivatkozik.

[6]Hypertext Transfer Protocol - Alkalmazási réteg protokoll, amely a kliens-szerver kommunikációt határozza



5. ábra. Az Athletimeter szoftverrendszer kommunikációs diagrammja.

szerint valósul meg. Minden függvényalkalmazás a saját REST[7] API[8]-ján [54] keresztül kommunikál a kliensekkel. Következtetésképpen egy állapotmentes adatcsere biztosított, az erőforrások szabványos formában (JSON formátumban) kerülnek továbbításra, és jól meghatározott URL-konvenciókon keresztül érhetőek el, továbbá a HTTP kérések és válaszok fejlécei megfelelő módon reprezentálják a végrehajtandó műveletet és ennek eredményét.

Az új eredmények valós idejű továbbítását a weboldalra a Pusher Channels API [27] valósítja meg, amely *publish-subscribe* modellre épül. Amikor az eredmények függvényalkalmazása létrehozó, törlő vagy módosító műveletet hajt végre, a szerver egy megfelelően elnevezett esemény formájában teszi közzé az információkat a csatornán. Amennyiben a kliensoldal ugyanarra a csatornára van rácsatlakozva, fogadni tudja ezeket az eseményeket, és így nem kell az információs oldalon adatlekérést végrehajtson a szinkronizálás érdekében.

A bemutatott kommunikáció, valamint a szoftverrendszer komponenseinek architektúrája a 2. fejezetben kerül részletes ismertetésre.

---

meg.

[7]Representational State Transfer - Architekturális minta, amely konvenciókat határoz meg kliens és szerver közti kommunikációra.

[8]Application Programming Interface - Szabályok összessége különböző szoftverkomponensek integrációjára vonatkozóan.

2. Az alkalmazás felépítése

2.1. Alkalmazásszerver

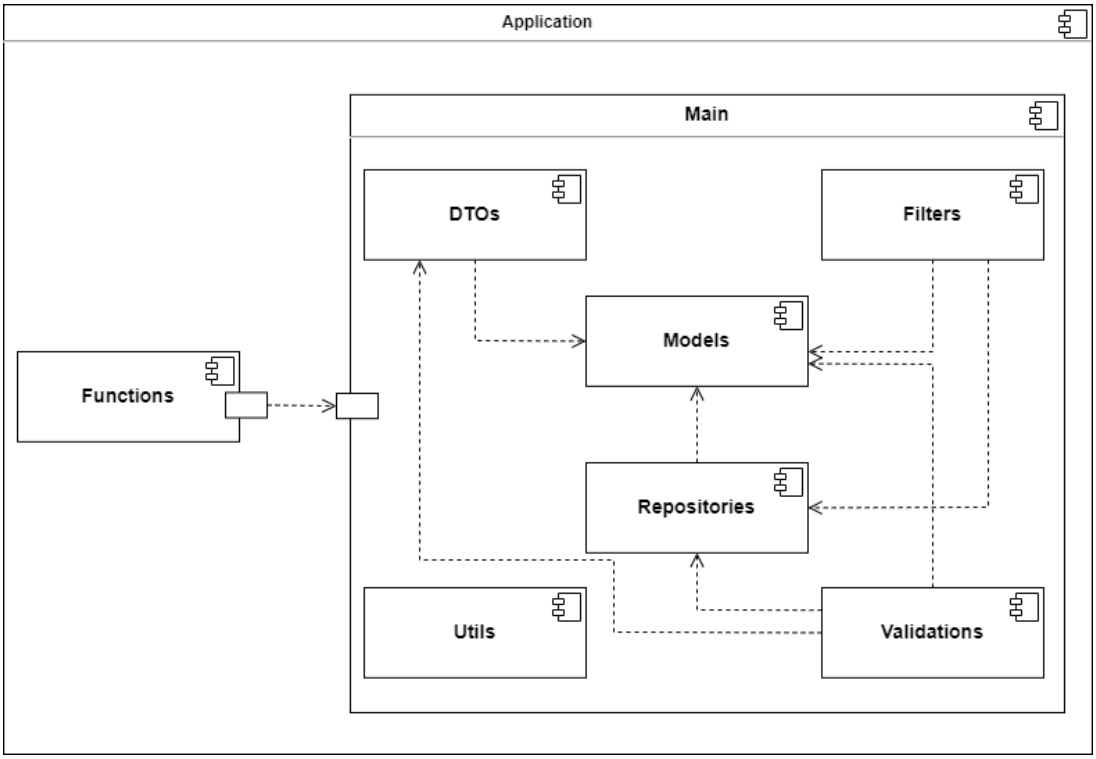
Az alkalmazásszerver egy hátteret biztosít a kliens alkalmazásoknak, feldolgozza a bejövő kéréseket és végrehajtja a szükséges műveleteket.

A szervert a *Microsoft Azure* felhőszolgáltató, és azon belül az *Azure függvények* biztosítják.

2.1.1. Szerver struktúrája

Az alkalmazásszervert két fő komponensre lehet osztani:

1. az *application* rész tartalmazza a függvényeket, a hozzájuk tartozó erőforrásokat, a modelleket és a hozzájuk tartozó DTO[9]-kat, az adatbázishoz történő kapcsolódáshoz szükséges erőforrásokat, illetve a műveletekhez tartozó segédosztályokat, függvényeket
2. a *configuration* rész tartalmazza az adatbázist létrehozó és az azt bekonfiguráló parancsokat



6. ábra. Az alkalmazás backend részének felépítése, a komponensek kapcsolatai

[9]Data Transfer Object - A modell különböző reprezentációját képező objektum, amelyet folyamatok közötti adatátvitelre használnak.

Ahogy a 6. ábrán látható, az application rész két fő komponensre tagolódik, a *Functions* és *Main* komponensekre, amelyek további alkomponenseket tartalmaznak.

Az alkalmazásban négy elkülöníthető entitás található: versenyző, szervező, eredmény, versenyszám. Mind a négy entitásért külön Azure függvényalkalmazás felel, ami tartalmazza az adott entitáshoz tartozó műveleteket. Ezenkívül a szervezők kivételével minden entitásnál szerepel egy PDF generáló, JSON-t létrehozó, és azt exportáló függvény is. A négy entitás kezelésén kívül, a felhasználók bejelentkezését is egy külön Azure függvény látja el. A korosztályok kezelését megvalósító műveletekre nincs külön függvényalkalmazás definiálva, ezek elérése a szervezők, versenyzők, illetve versenyszámok függvényén keresztül lehetséges.

Az alkalmazásban implementálva van a *Factory*[10] és *Abstract Factory*[11] tervezési minta, amelyek - többek között - az adathozzáférésért felelős komponensek cserélhetőségében játszanak szerepet. A második nagyobb komponens a *Main*, ahol azok a közös erőforrások, kódok találhatóak, amiket minden függvény használ.

1. **Models**: itt található minden entitásnak a reprezentációja a különböző tulajdonságaikkal együtt. Emellett itt található az adatbázist konfiguráló, és az adatbázis kapcsolatot reprezentáló objektumot visszaadó factory is.

2. **DTOs**: a nagyobb biztonság és az átküldendő adatmennyiség minimalizálása érdekében a teljes modellek helyett, DTO-k használatával, ezek különböző reprezentációi kerülnek elküldésre vagy lekérésre. Az átalakítást a modellből DTO-ba, és fordítva, mapper-ek végzik.

3. **Filters**: külön komponenseket képeznek a szűrők, amelyeket entitások lekérésekor lehet megadni. Ezek a függvények az adott entitásokból azokat térítik vissza, amelyek megfelelnek az adott feltételeknek.

4. **Repositories**: itt találhatóak a különböző entitások CRUD[12] műveleteit implementáló osztályok, két szinten is absztrakciót használva, úgy interface, mint generikus paraméter szintjén, így növelve az általánosságot és újrahasználhatóságot.

5. **Utils**: ebben a komponensben olyan általános műveletek találhatóak, amelyek nem tartoznak egyik entitáshoz sem, mint például a jelszó hashelés, JWT token generálása,

[10]Factory - olyan tervezési minta, aminek segítségével úgy tudunk példányosítani objektumokat, hogy nem kell tudjuk, milyen implementéció van a háttérben, csak a közös interfészt kell ismernünk

[11]Abstract factory - Factory tervezési minta továbbgondolása, itt már a Factory példány is cserélhető

[12]Create, Read, Update, Delete - A 4 alapvető adatkezelési művelet.

Pusher rendszer létrehozása és konfigurálása stb.

6. **Validations**: ez a komponens tartalmazza az entitásokat validáló függvényeket, amelyek leellenőrzik, hogy jó adat érkezik-e meg a szerverhez, és hiba esetén a megfelelő üzenettel térnek vissza.

Az alkalmazásszerver másik fő része a *configuration* komponens. Ez a komponens felel az adatbázis létrehozásáért, konfigurálásáért. A táblák és a köztük levő kapcsolatok felépítését is ez végzi, egy megadott *.sql* kiterjesztésű fájlból véve a parancsokat.

2.1.2. Szerver architektúrája

Az alkalmazásszerver, ahogyan az a 2.1.1. fejezetben említésre került, Azure függvényeket használ, amelyek egy REST API-t szolgálnak ki. Minden függvényalkalmazáson belül több függvény található, amelyek különböző útvonalakra figyelnek, és várják a kéréseket. Ezek az útvonalak megfelelnek a REST szabványnak.

Ezenkívül az Azure függvények egy *szerver nélküli (serverless)* architektúrát biztosítanak, aminek lényege, hogy a fejlesztő feltölti a kódját a felhőbe, megadva bizonyos konfigurációkat (mikor hívódjon meg az adott kód, a skálázást, az erőforrásokat), és a kéréseknek a továbbítását a felhőszolgáltató biztosítja, a konfigurációban megadott esemény bekövetkeztekor meghívva a programozó által feltöltött kódot.

Így a fejlesztésben csak a kódra lehet fókuszálni, nem szükséges azzal foglalkozni, hogy hogyan jut el a kérés a lekezelő függvényig, mivel ezt megoldja az Azure.

Az Azure függvények használatának egy másik előnye, hogy nincs szükség állandó szerverre, amely folyamatosan erőforrásokat használ, ehelyett a függvények csak akkor használnak erőforrásokat, amikor valamilyen kérés érkezik hozzájuk. Mivel a felhőben rendelkezésre álló szolgáltatások nincsenek ingyen, ez azért is előnyös, mert így csak azokért szükséges fizetni, amelyek ténylegesen használódnak. Egy hátrány az indítási idő (cold start time), ami miatt az első kérés lassabban lesz kiszolgálva.

Az alkalmazás csak a verseny napján van kitelepítve, más napokon nem használ erőforrásokat az Azure-on, így alacsonyak maradnak a költségek.

Az Azure a programozó által megadott konfiguráció alapján dönti el, hogy melyik Azure függvénynek kell továbbítania a kérést. A függvények egy HTTP objektumot kapnak paraméterként, és egy loggert, amivel log üzeneteket tudunk megadni. Ezenkívül meg kell adni a HTTP kérés típusát (GET, POST, DELETE, PUT, PATCH) és az útvonalat is.

2.1.3. Szerverhez érkező kérések feldolgozása

A szerverhez érkező kéréseket, ahogyan az a 2.1.2. fejezetben is elhangzott, a serverless architektúrának köszönhetően, az Azure felhőszolgáltató juttatja el a megfelelő függvényekhez, amikor érkezik egy HTTP kérés.

Ezeket a kéréseket a függvények *HttpRequest* objektumok formájában kapják meg, ami tartalmaz minden információt, amire szükség lehet a kérés kiszolgálásához.

Kezdetben a legtöbb függvény leellenőrzi, hogy a megfelelő jogosultsággal van-e ellátva a kérés, pontosabban, hogy bejelentkezett és megfelelő szerepkörrel rendelkező felhasználótól jött-e a kérés. Ezek a függvények a 2. ábrán feltüntetett funkcionalitásoknak felelnek meg.

Ezt követően a bejövő DTO-k átalakulnak a megfelelő modell objektumokba, megtörténik az adatok validálása és a válasz felépítése, figyelve arra, hogy minél kisebb adatmennyiség kerüljön mozgatásra.

Miután a válasz felépül, egy *IActionResult* objektum térül vissza, amit az Azure HTTP válasszá alakít, és azt küldi vissza a böngészőnek.

A hibák kezelésénél, minden hibánál, legyen az validálási, szerver oldali, vagy kliens oldali hiba, mindig a megfelelő HTTP válasz kerül visszaküldésre, ami tükrözi azt, hogy mi történt, figyelve arra, hogy ne jusson ki a kliens oldalra szerver-specifikus információ.

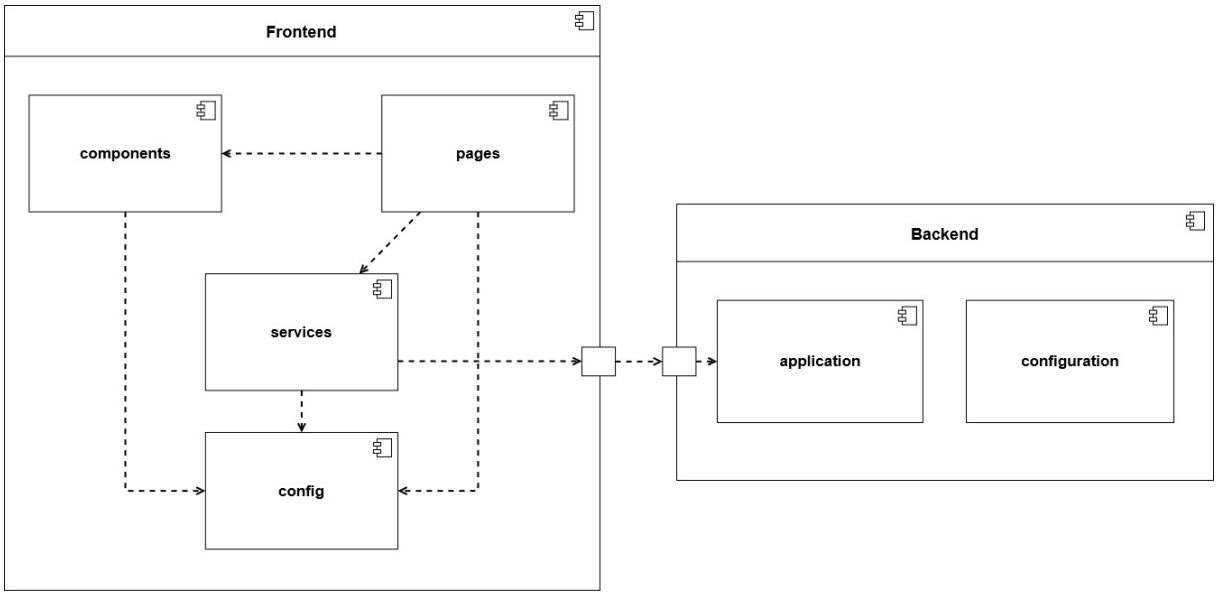
2.2. Web kliens

A szoftverrendszer webalkalmazása egy könnyen kezelhető platformot biztosít a szerver által szolgáltatott adatok kezelésére és megtekintésére. Ezt a *React.js* keretrendszer komponens alapú technológiájának alkalmazása teszi lehetővé.

2.2.1. A kliens struktúrája

A weboldal struktúráját négy fő komponensre lehet bontani, amelyek a 7. ábrán levő diagrammon láthatóak. Ez a felépítés könnyen kezelhető és átlátható modulokra bontja a projektet.

A *components* réteg magába foglalja az oldal fejlécét, a navigációt megvalósító oldalmenüt, illetve azokat az elemeket, amelyeket a fő konténer által megjelenített, *pages*-ben levő komponensek használnak fel. A *components* emellett felhasználja a *config* csomagban előre definiált konstansokat, valamint a nemzetköziesítést a fejlécen levő nyelvcsere biztosítása



7. ábra. A frontend felépítése a komponensek közötti függőségi kapcsolatokkal.

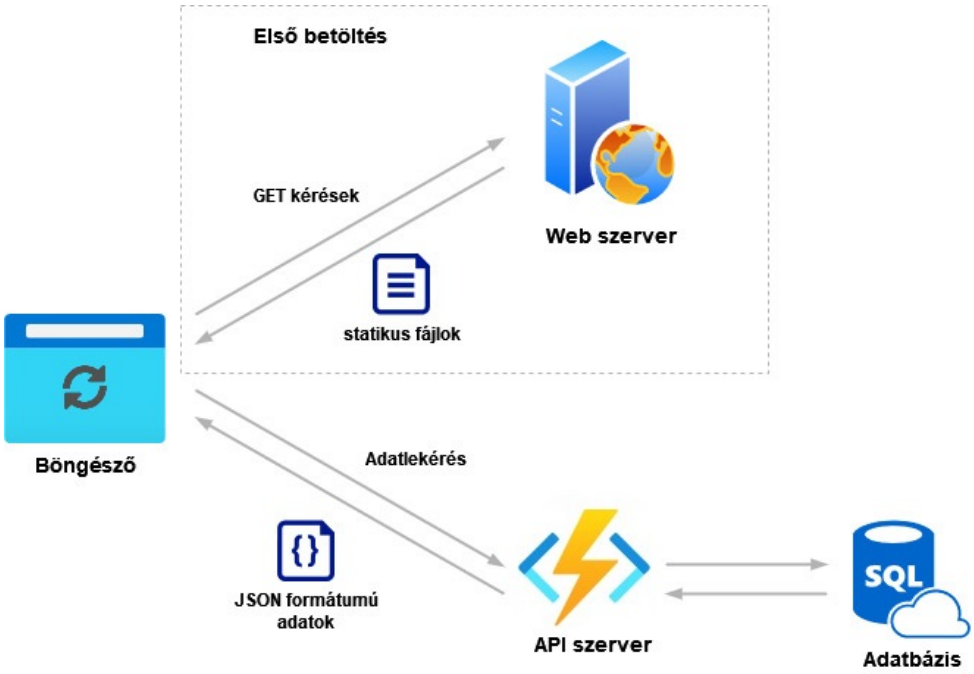
érdekében.

A *pages* modulban szerepel a frontend fő konténere, amelynek tartalma az oldalmenün kiválasztott opció alapján változik. Az oldalak, amelyeket a fő konténer megjelenít, egy-egy almodulban foglalnak helyet. Ezek magukba foglalják a főoldalt, a bejelentkezési oldalt, a profiloldalt, a CRUD műveleteket szolgáltató felhasználói felületeket, illetve az információs oldalt, amely a fő konténerrel egy szinten helyezkedik el a komponenshierarchiában. A modul függ a *service* rétegtől, az oldalak állapotai a megfelelő *service* metódusok hívása alapján frissülnek.

A *service* könyvtár felelős a szerverrel való kommunikációért, amely a *Fetch API* segítségével valósul meg. Az API lehetővé teszi a modulban definiált függvények számára az erőforrások aszinkron lekérését. (további részletek a 2.3.2. fejezetben).

A *config* csomag fájljai az újrahasznosítás és a kódismétlés elkerülése érdekében, olyan előre definiált konstansokat foglalnak magukba, mint például a lekérdező paraméterek vagy az elérési útvonalak, amelyeket a *pages* komponensei, illetve a *service* metódusai használnak fel. Emellett a modul tartalmazza a nemzetköziesítést megvalósító fájlokat is.

A webalkalmazás stílusbeállításai az *assets* csomagban kapnak helyet, illetve egy külön *icons* modulban találhatóak a platform kinézetét elősegítő képek és ikonok.



8. ábra. SPA szemléltetése

2.2.2. Architektúra

Az alkalmazás az MVC[13] [10] tervezési mintára épül. Az 1.2. fejezetben bemutatott kommunikációs modell alapján a weboldal REST API-on keresztül kommunikál a szerverrel. Az aszinkron HTTP kérést a paraméterként megadott webcímnek megfelelő API kezeli le. A szerver kizárólag az adatok szolgáltatásáért és kezeléséért felel, a válaszként kapott információk, illetve a felhasználói felületen kiváltott események alapján a megjelenítés frissítését a kliensoldal hajtja végre.

Ezek alapján elmondható, hogy az alkalmazás *rich client* [5] architektúrával rendelkezik, hiszen az MVC mintán belül definiált megjelenítési és vezérlési komponensek műveletei a böngésző oldalán kerülnek végrehajtásra.

Mint ahogyan az előző fejezetben említve volt, a webkliens egy *MainContainer* nevű komponenssel rendelkezik, amely az oldalmenün kiválasztott opció alapján jeleníti meg a hozzárendelt elérési útvonalnak megfelelő *pages* elemeket. Ezek alól az információs oldal kivételt képez, ugyanis ennek az oldalnak, hasonlóan a főkonténerhez, az *App* komponens a szülője a DOM[14] hierarchiában, amely különböző elérési útvonalakhoz rendeli a két elemet.

Következtetésképpen a webkliens egyetlen oldalból áll, így SPA[15] [12] típusú

[13]Model-View-Controller - Tervezési minta, amely a megjelenítés és vezérlés szétválasztására nyújt megoldást.
[14]Document Object Model - Objektumok ábrázolása, amely egy webes dokumentum szerkezetét szemlélteti.
[15]Single-Page Application - Webalkalmazás-típus, amelyen belül a tartalom egyetlen oldal frissítésével változik.

alkalmazásnak tekinthető. Mint ahogyan a 8. ábra is szemlélteti, a böngésző kizárólag az első betöltéskor kéri le a webszervertől az alkalmazás egyetlen oldalát képező HTML fájlt, illetve a hozzá tartozó statikus CSS, JavaScript állományokat és képeket. Nincs teljes oldal újratöltés, ezután csak azokat a komponenseket rajzolja ki újra az oldalon, amelyek esetén változást észlelt. Ezáltal a felhasználók számára egy gördülékenyebb és élvezhetőbb platform biztosított.

2.2.3. Komponensek

A webalkalmazáson belül meghatározott komponensek a *components*, illetve a *pages* modul elemei. A felhasználói felület egymástól független, kisebb részekből tevődik össze. Az újrahasznosítás és hatékonyság érdekében, azok a komponensek, amelyek több, különböző oldalon is megjelennek, mint például a listázó oldalakon szereplő exportálást megvalósító gombok, külön elemként definiáltak. Emellett a bonyolultabb struktúrával rendelkező oldalak egyszerűbben kezelhető komponensekre vannak felbontva, hogy egy átlátható és könnyen kezelhető kódbázis legyen biztosítva.

A komponensek *.jsx* kiterjesztésű állományokban foglalnak helyet, amelyek lehetővé teszik HTML kód írását JavaScriptben. Ez azt jelenti, hogy ugyanabban a fájlban szerepelhet HTML és JavaScript kód is, és ezek együttesen írnak le egy React elemet. A böngészők viszont nem képesek ezt a formátumot feldolgozni, ezért a React a Babel [3] nevű fordítót használja annak érdekében, hogy a *.jsx* állományok JavaScript kóddá alakuljanak át. A háttérben a fordító a HTML kódot a böngésző által megértett *React.createElement* [46] függvény meghívásával helyettesíti, amelynek paraméterként megadja a HTML elemet definiáló információkat.

Az elemek függvénykomponensként [43] vannak definiálva. Ezek tulajdonképpen olyan JavaScript függvények, amelyek React elemeket térítenek vissza. A függvények argumentuma lehet a *props* objektum, amely a külső komponensek által átadott tulajdonságok elérésére szolgál. Az osztálykomponensekkel ellentétben, függvények esetén nem létezik *state* objektum, amelynek módosítása maga után vonja a komponens újra renderelését. Ebben az esetben az állapotváltozók komponenshez való rendelése horgok (hooks) használatával van megvalósítva.

2.2.4. Horgok

A React 16.8-as verziójában bevezetett *Hooks API* [47] lehetővé teszik a komponensek állapotának illetve életciklusának kezelését osztálydeklaráció nélkül. Segítségükkel a webkliens komponenseit olyan függvények együtteseként lehet definiálni, amelyek mindegyikének

megvan a saját feladata az állapotkezelési logika végrehajtásában.

Az állapotváltozók függvénykomponenshez való rendelését a *useState* [53] horog biztosítja. Egy olyan párost térít vissza, amely a kezdeti értékkel ellátott állapotváltozóból, illetve az állapot frissítését megvalósító függvényből áll. Az állapot értékének felülírása, a függvény meghívásán keresztül, a komponens frissítését eredményezi.

Komplex állapotlogika esetén a *useReducer* [52] horog van alkalmazva, amely megkönnyíti objektumállapotok esetében az alértékek módosítását. A kezdeti érték mellett, paraméterként egy *reducer* függvény is adott. A metódus egy *tiszta függvény* [40], ugyanis a végrehajtás során nem lépnek fel mellékhatások, és ugyanazokra a bemenetekre mindig azonos kimenetet térít vissza. Feladata az eddigi állapotból és a horog által visszatérített *dispatcher* metóduson keresztül átadott *action* objektumból egy új állapot visszatérítése, amelynek hatására megvalósul a komponens újrarenderelése.

A 1. kódrészlet egy példát szemléltet a *useReducer* horog használatára. A versenyszámokhoz tartozó listázó oldal állapotát, illetve az ehhez tartozó *reducer* függvényt foglalja magába. Ezen belül három művelettípus van meghatározva, amelyek az oldal betöltésekor fellépő lehetséges fázisokat jelképezik, azaz a betöltésre való várakozást, a sikeres

```
const initialState = {
  events: [],
  isLoaded: false,
  error: null,
};

function reducer(state, action) {
  switch (action.type) {
    case 'loadEvents':
      return { isLoaded: true, events: action.payload, error: null };
    case 'loadError':
      return { ...state, isLoaded: true, error: action.payload };
    case 'load':
      return { ...state, isLoaded: false };
    default:
      throw new Error();
  }
}

const [data, dispatch] = useReducer(reducer, initialState);
```

1. kódrészlet. Példakódrészlet, amely a useReducer horog definícióját szemlélteti.

adatlekérést, illetve a hiba fellépését. A *dispatcher* függvény meghívásán keresztül valósítható meg az állapot alértékeinek a módosítása, megadva paraméterként az *action* objektumot, amely konvenció szerint két mezővel rendelkezik: a *type* mező a művelet típusát jelképezi, a *payload* pedig azokat az értékeket foglalja magába, amelyekkel az új állapot fog rendelkezni.

Az osztálykomponensekkel ellentétben, a komponensek funkcionális deklarációja nem nyújt előre definiált függvényeket a különböző életciklus állapotok kezelésére. Ennek megvalósítására a *useEffect* [50] horog kerül felhasználásra, amelynek segítségével mellékhatásokat lehet végrehajtani. Első paramétere egy *callback* függvény, amely az elem kirajzolása után kerül végrehajtásra, emellett opcionálisan megadható egy tömb paraméter, amely a hatás függvény függőségeit foglalja magába.

Ha a horog nem rendelkezik ezzel az argumentummal, akkor a függvény minden komponensfrissítéskor lefut, mint ahogyan osztályok esetében a *componentDidUpdate* [42] metódus.

Üres tömb esetében a *callback* függvény a *componentDidMount*-tal [41] egyenértékű, azaz kizárólag a legelső kirajzoláskor hajtódik végre. A webalkalmazásban ez olyan esetekben van alkalmazva, amikor az aktuális oldal szervertől kapott adatokat kell megjelenítsen, mint például az adatmódosító oldalak vagy a profiloldal esetén.

Ha a tömb tartalmaz állapotváltozókat, akkor a mellékhatás csak akkor kerül végrehajtásra, ha a felsorolt változók valamelyike módosult. Például listázó oldalak esetén, amikor a szűrési lehetőségeket tartalmazó állapot módosul, az adatok újra le lesznek kérve a szervertől, a beállított szűrőknek megfelelően.

Bizonyos mellékhatások lefutása után szükséges lehet takarító függvények végrehajtása, amelyek a komponens DOM-ból való eltávolításakor futnak le. Osztálykomponensek esetében ez megegyezik a *componentWillUnmount*-tal [44], ezzel szemben a horog használatakor ez a *callback* függvény visszatérítési értéke lesz, amelyet a React tisztítás során használ fel. Abban az esetben, amikor a weboldalon oldalváltás történik, még mielőtt válasz érkezett volna a szervertől, a *fetch* kérés megszakad, ezt pedig az *AbortController* [6] *signal* értékén keresztül jelzi, ezáltal biztosítva a felesleges kérések megakadályozását, illetve a nem létező elemre való hivatkozás miatt fellépő hibák elkerülését.

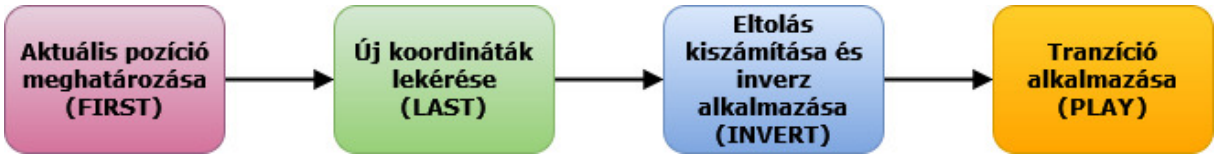
2.2.5. Élő frissítés

A *Pusher Channels API* által szolgáltatott csatorna lehetővé teszi, hogy a weboldal valós időben kapjon értesítést a versenyszámok során kialakult eredményekről. Az eredményeket kezelő függvényalkalmazás különböző típusú események formájában jelzi a csatornán egy új eredmény elmentését, egy létező módosítását, illetve inaktívvá változtatását. Az első betöltéskor a kliensoldal lekéri a kiválasztott eseményhez tartozó összes eredményt, ezután pedig az új információkat a csatorna *bind* [28] metódusán keresztül fogadja, amelynek paraméterei az esemény neve, illetve egy *callback* függvény, amelyen belül az események adatai alapján hajtja végre a lista megfelelő frissítését megvalósító metódusokat.

Például, amikor egy új eredmény kerül be a rendszerbe, a függvényalkalmazás, miután elmentette az új adatot, egy *'refresh'* eseményt tesz közzé a Pusher csatornán, amely az új eredményhez tartozó információkat foglalja magába. A kliensoldal fogadja ezt az eseményt, majd a kiolvasott adatot elhelyezi az eredmények rendezett listájában, a megfelelő helyre.

Az eredmények frissítésekor a felületen egy lista-átrendezést szemléltető animáció megy végbe, amelynek célja az információs oldal dinamikájának elősegítése, és a felhasználói élmény növelése. Ennek megvalósítása a FLIP [16] [25] technikára épül. A 9. ábrán látható négy lépést határozza meg, amelyek közvetlen módon hajtanak végre DOM-módosítást az elemeken. A *useLayoutEffect* [51] horog lehetővé teszi ezek szinkron módon történő végrehajtását a legelső kirajzoláskor. Az első lépés az eredményeket szemléltető *ResultCard* komponensek aktuális pozíciójának meghatározásából áll. Az eredmények frissítése után az elemek helyzete megváltozhat, második lépésben az új koordináták kerülnek meghatározásra a JavaScript által nyújtott *getBoundingClientRect* [8] metódus segítségével. Ilyenkor a DOM hierarchiában a *card* elemek már az új pozíciókban foglalnak helyet, és ezeknek az értékeknek az ismeretében tudni lehet, hogy a komponensek mennyit és milyen irányba kerülnek majd elmozdulásra a következő renderelésnél. Ahhoz, hogy az átrendezést vizualizáló animáció végbe tudjon menni, szükséges megakadályozni, hogy a böngésző automatikus pozícióváltása hajtódjon végre újrarajzolásnál. Ezért a következő lépésben a kiszámolt eltolások inverze kerül alkalmazásra, és így a böngésző számára az elemek látszólag az eredeti helyükön maradnak. A *requestAnimationFrame* [13] függvény közvetlenül az újrarajzolás előtt kerül végrehajtásra, az utolsó lépésben a neki paraméterként megadott metódus, egy megfelelő tranzíció beállításával, befolyásolja az elmozdulások megjelenését a felületen.

[16] First-Last-Invert-Play - Lista-átrendezés animálását megvalósító princípium.



9. ábra. FLIP technika lépései

A felhasználók egy oldalon maximálisan nyolc eredményt láthatnak. Ahhoz, hogy megtekinthessék a teljes mezőnyt, lehetőségük van az oldalak között lapozni. A legfrissebb eredményekről való lemaradás elkerülése érdekében, abban az esetben, ha az új eredmények nem a felhasználó által látott oldalra kerülnek, a rendszer az információkat értesítések formájában közli.

Az értesítések kezelése egy saját *useNotification* Horog használatával valósul meg. Ennek egyetlen paramétere egy milliszekundumban kifejezett érték, és megadja, hogy ezek mennyi ideig lesznek láthatóak a felhasználó számára. Egy függvényt térít vissza, amely az új értesítések létrehozására, illetve a megadott időtartam eltelte után ezek automatikus eltávolítására szolgál. Az értesítésekhez tartozó állapotváltozók kezelése egy *NotificationProvider* komponensen belül történik és ez a React által nyújtott *Context API* [45] használatára épül, amely az állapotok megosztásának megkönnyítésére szolgál. Az értesítések kezelését megvalósító függvény és az értesítéseket jelképező állapot nem *props* formájában jut el a horoghoz, illetve a értesítéseket tartalmazó konténerhez. A *NotificationProvider* komponens két egymásba ágyazott *Provider* elemből áll, amelyeken keresztül ezeket *context* objektumok formájában teszi közzé. A komponensek pedig *useContext* horgon keresztül kérik le a megfelelő *Provider*-től a számukra szükséges objektumot.

2.2.6. Biztonság

Az 1.1 fejezetben említve volt, hogy az alkalmazásban több különböző szerepkör van, és ezek különböző funkcionalitásokat, oldalakat érnek el.

A web kliens esetében három szerepkör van megkülönböztetve: adminisztrátor, szervező és vendégfelhasználó. Minden szerepkör esetén kizárólag a megfelelő oldalak és funkcionalitások érhetőek el, így szükséges annak ellenőrzése, hogy egy felhasználó be van-e jelentkezve, és ha igen, akkor milyen szerepkörrel rendelkezik.

A web kliens először elküldi a felhasználónév-jelszó párost a szervernek, amely alapján a szerver leellenőrzi, hogy a megadott értékpáros helyes-e, és ha igen, akkor sikeresnek tekinti a bejelentkezést. Egy JSON Web Token [21] generálódik a felhasználónak, amely a

felhasználónevét, az azonosítóját és a szerepkörét tartalmazza, kulcs-érték párosokkal megadva, és titkosítva. A szerver a három értéknek és a generált token-nek megfelelően egy objektumot hoz létre, amelyet a web kliensnek küld el. A web kliens megkapja ezt az objektumot, és a token-t egy böngésző sütiben (browser cookie) [7] tárolja el. A süti a *sameSite* flag használatával véd a CSRF támadások ellen is. Ez a flag mondja meg azt a böngészőnek, hogy hogyan és mikor küldjön sütiket, elsődleges vagy third-party (külső) szituációkban. Ha ezt a flage-t *strict* értékkel van ellátva, akkor nem fog küldeni sütiket a kéréssel, ha az a kérés egy másik oldalról érkezett. Vagyis, ha egy másik oldalról lenne elküldve ugyanaz a kérést, akkor a böngésző nem fogja hozzá társítani a sütit. Kijelentkezéskor ez a süti törlődik a böngészőből, és a felhasználó újra vendégnek tekinthető.

2.3. Mobil kliens

A mobil kliens a szerverrel kommunikál, és egy felhasználói felületet üzemeltet Android és iOS rendszereken. A *React Native* technológia lehetőséget ad arra, hogy egyetlen kódbázis feleljen a megjelenítésért mindkét operációs rendszer esetében. A React primitívek natív platform-specifikus felhasználói felületként jelennek meg, ezzel gyorsítva a mobilalkalmazások fejlesztési folyamatát. Bizonyos komponensek más megjelenítést kapnak a különböző platformokon, így jobban beleillenek a platform esztétikai elvárásaiba. Az alkalmazás elsődleges célplatformja az Android.

2.3.1. A mobil kliens struktúrája

Az alkalmazás kiindulópontja a gyökér mappában található *App.js* fájl. Emellett az alap mappában megtalálható a *package.json* fájl, amit minden npm által menedzselt projekt tartalmaz. Ez tárolja a projekt meta adatait, például a használt függőségeket. Az *app.json* fájl konfigurációs kulcs-érték párosokat tartalmaz a projektre vonatkozóan, mint az applikáció neve, ikonja, verziója.

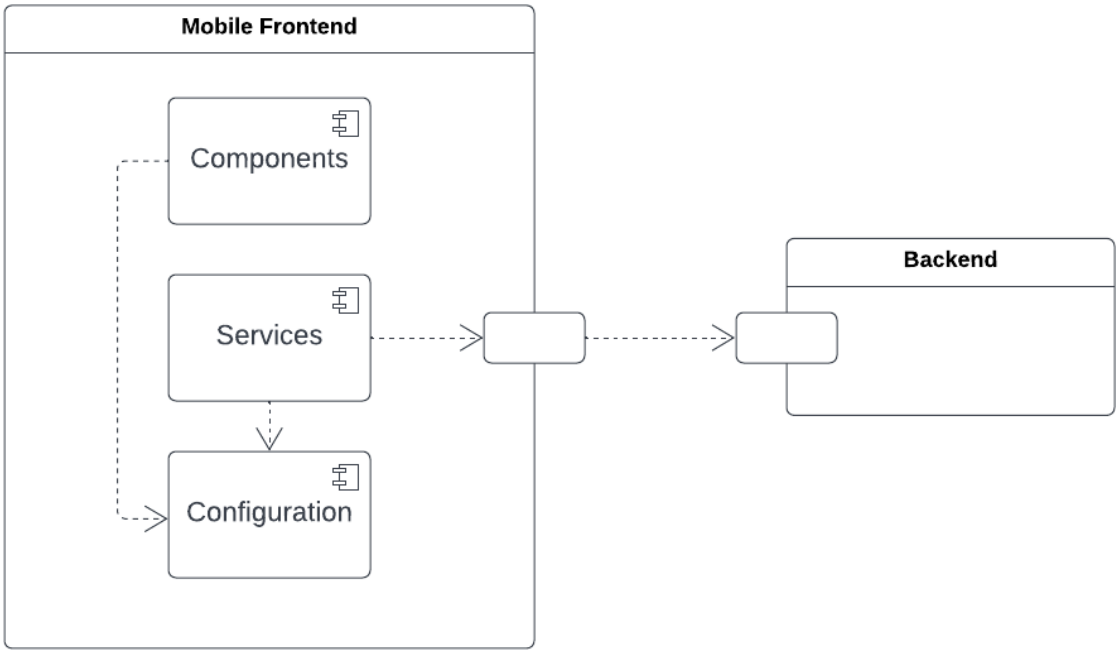
A *components* mappában található függvénykomponensek építik fel a megjelenítendő oldalakat. A függvények React Native-ban, a React-hoz hasonlóan valid komponensnek számítanak, mivel bemeneti paraméterként egyetlen *props* tulajdonságokat tartalmazó változót kapnak, majd React elemeket térítenek vissza.

A *services* könyvtár kommunikál a szerverrel. Itt három különböző *.js* kiterjesztésű fájl felel a különböző entitásokhoz tartozó fetch műveletek elvégzéséért. A három entitás megfeleltethető

a szerveroldali Azure függvényekkel, egyenként felelnek a versenyzőkhöz, versenyszámokhoz, illetve eredményekhez tartozó műveletekért. Függvényeket tartalmaznak, melyek fetch kérések által lekérik vagy frissítik a már meglévő adatokat, vagy újakat szúrnak be.

A *config* mappában található konstansok segítségével adódik meg az az IP cím, melyen elérhetjük a felhőben tárolt Azure függvényeket, valamint az *ip18Config.js* fájl segítségével itt valósul meg a globalizáció, jelenleg a felhasználók az angol és magyar nyelv között választhatnak. A *locales* almappában találhatóak az ezekhez a nyelvekhez szükséges fájlok, melyek tartalmazzák az alkalmazás szöveges tartalmait az adott nyelven.

Az *assets* mappában megtalálhatóak az alkalmazás által használt statikus erőforrások, képek ikonok.



10. ábra. A mobil alkalmazás felépítése

2.3.2. Kommunikáció a szerverrel

Az alkalmazás a *Fetch API* [9] segítségével kommunikál a szerverrel. A Fetch általános *Request* és *Response* objektumokat szolgáltat, melyek segítségével elérhető a szerver oldali REST API. Fontos tulajdonsága, hogy ígéretekkel dolgozik visszahívások helyett, így aszinkron módon dolgozhatók fel a kérések. A *Promise* [11] objektum proxy egy olyan értékre, mely nem feltétlen ismert a meghívás pillanatában. Egy várakozásban levő ígéret, vagy teljesül, vagy

hiba esetén elutasítodik, ilyenkor a megfelelő *then* vagy *catch* ág kerül meghívásra. A fetch metódus első paramétere egy URL, a második pedig egy JSON objektum, ami tartalmazza a kérés fejlécét, típusát és a kérés törzsét. Válaszként egy JSON objektum lesz visszatérítve.

2.3.3. Navigáció

A navigálást az oldalak között a React Navigation [38] közösségi csomag biztosítja. Az App.js fájlban beburkoljuk az alkalmazás többi kódját egy *NavigationContainer* komponensbe, és konfiguráljuk egy *Drawer.Navigator* tulajdonságként a kezdeti elérési útvonalat, majd *Drawer.Screen*-ek tulajdonságaként, az összes többi szükséges útvonalat. A meglátogatott oldalak egy verembe kerülnek, melyben kezdetben csak az alapértelmezett oldal útvonala található meg, majd minden *navigation.navigate('route')* hívás után az adott fájl elérési útvonala is bekerül a verem tetejére. Ennek köszönhetően visszalépéskor könnyen be tudjuk azonosítani, melyik oldalra kell visszatérnünk.

2.3.4. Biztonság

A mobil kliens csak a szervezők számára elérhető, nem érhető el semmilyen funkció mobilon, anélkül, hogy a felhasználó bejelentkezne.

A 2.2.6. fejezetben bemutatott módszerhez hasonlóan, a platform elküldi a szervernek az adatokat, és ugyanolyan választ kap vissza, mint a web kliens. Azonban, a webes hitelesítéssel szemben, mobilon a *SecureStore* [17]-ban tárolódik el a szervertől kapott felhasználónév, azonosító, szerepkör és az ezek alapján kigenerált token.

3. Technológiák és eszközök

3.1. Azure felhőszolgáltatás

Ahogy az a 2.1. alfejezetben is említve volt, a szerver implementálásához a *Microsft Azure* [36] felhőszolgáltató volt használva.

A Microsoft Azure egy olyan platform, amely felhőszolgáltatások segítségével megkönnyíti a fejlesztést. 2010 februárja óta elérhető [57] teljes körben, és azóta nagy fejlődéseken ment át. Sok technológia bekerült, mint a nagy adat (Big Data) támogatás, az intelligens felhő (Inteligent Cloud), vagy a serverless architektúrát megvalósító Azure Functions. Tárolhatóak rajta különböző típusú adatbázisok is, emellett használhatóak a docker, illetve kubernetes [1] konténerek is. Kérhetőek és futtathatóak virtuális gépek, a felhasználó határozhatja meg milyen konfiguráció kell neki, és annak függvényében fizeti az erőforrásokat.

3.1.1. Összehasonlítás más felhőszolgáltatókkal

A Microsoft Azure mellett, a másik két nagy felhőszolgáltató az Amazon Web Services (AWS) és a Google Cloud. Mindhárom szolgáltatónál megtalálhatóak a főbb szolgáltatások (adatbázisok, különböző serverless technológiák, különböző tárhelyek), és emellett mindhárom szolgáltatónál megtalálható valamennyi mesterséges intelligencián alapuló eszköz is. A különbség a szolgáltatások számában, az árazásban, és az elterjedésben rejlik.

A piac legnagyobb részét az AWS fedi le a maga 32 százalékával, ő a legrégebbi szolgáltató, 2006-ban jelent meg. A legtöbb szolgáltatással rendelkezik, több mint 200 vehető igénybe ezekből [22].

A Google Cloud egy fiatalabb szolgáltató, 2011-ben jelent meg [22], és egyre több szolgáltatás elérhető rajta. Nagy előnye az árazása [23], felhasználóbarát, és különböző kedvezmények is igénybe vehetőek.

Az Azure több szolgáltatással rendelkezik, mint a Google Cloud, viszont kevesebbel mint az AWS. Nagyon jól integrálva vannak benne más Microsoft, illetve nyílt forráskodú programok, amiket nagyobb vállalatok előszeretettel használnak. Ilyen szempontból, ha Microsoft technológiákkal van fejlesztve egy alkalmazás, akkor az Azure a legkézenfekvőbb megoldás.

3.2. Szerver technológiák

ASP.NET Core [14] Az ASP.NET Core egy cross-platform[17], nagy teljesítményű, nyílt forráskodú keretrendszer, amely lehetővé teszi webalkalmazások fejlesztését. Lehetőséget nyújt az alkalmazások felhőbe való kitelepítésére, webalkalmazás szolgáltatások fejlesztésére, szerverek létrehozására. Mindezeket a **.NET Core keretrendszer** [35] segítségével lehet futtatni.

Azure Functions Core Tools [30] Az Azure Functions Core Tools egy eszköz, amellyel Azure függvényeket lehet karbantartani parancssorból. Így az Azure függvények fejlesztése, tesztelése, debuggolása, és a függvények kitelepítése lokálisan elvégezhető. Ezeket hozzá lehet rendelni a *Microsoft Azure* előfizetéshez, így saját környezetből is elérhetőek az Azure szolgáltatásai.

Entity Framework Core [32] Az Entity Framework Core egy kiterjeszthető, könnyű és nyílt-forráskodú objektum-relációs leképező (ORM[18]) rendszer. Sok adatbázist támogat: SQL Database, SQLite, MySQL, PostgreSQL és Azure Cosmos DB. Lehetővé teszi a leképezést mindkét irányba, azaz modellosztályokat lehet generálni az adatbázis táblák alapján, illetve a modellosztályokból egy új adatbázis felépítése is megvalósítható. Az adatokon való műveletek elvégzése (hozzáadás, törlés, módosítás) a modellosztályokon keresztül, objektumorientált kódból lehetséges, így nem szükséges SQL parancsok írása. Egy specifikus nyelv, a LINQ (Language Integrated Query) [33] biztosított, amely a modellosztályok attribútumain keresztül hajtja végre az utasításokat. Minden művelet egy kontextus objektummal végződik, amely az adatbázis kontextust teremti meg. Minden kontextus példánynak van egy *ChangeTracker* objektuma, amely nyomon követi az adatbázisba bekerülő változásokat. Ezeknek a változtatásoknak a szinkronizálása kódból is végrehajtható a kontextus *SaveChanges* függvényének meghívásával.

Azure Functions [29] Ahogyan ez a 2.1.2. fejezetben elhangzott, az alkalmazás szerverét Azure függvények képezik, amelyek egy serverless (szerver nélküli) architektúrát biztosítanak. Karbantartják a folyamatosan módosuló infrastruktúrát, és az erőforrásokat, így a programozó csak a kódra összpontosíthat.

JSON Web Token [21] A JSON Web Token (JWT) egy JSON objektum, és az adatok küldésének biztonságos módját teszi lehetővé. Az adatok a JWT-ben digitális aláírással

[17] Több platformon fut, pl. Windows-on, Linux-on, Mac-en

[18] Object-Relational Mapping - Egy programozási technika, ami lehetővé teszi egy objektum modell megfeleltetését egy relációs sémával.

rendelkeznek, így a benne levő információ ellenőrizhető és megbízható. A JWT-nek három fő része van *Header*, *Payload*, *Signature*, amelyek egy-egy ponttal elválasztva egymástól, egymásután összefűzve jelennek meg. A *Header*-ben általános információk szerepelnek, magába foglalja a kriptálási algoritmust (hash függvény legtöbbször) és a típust, base64URL kódolásra átírva. A *Payload* tartalmazza az elküldésre szánt tényleges adatokat, szintén base64URL kódolásra átírva. A *Signature* pedig az előző két rész egy ponttal elválasztott összefűzése, amely egy *secret*[19] felhasználásával van elhashelve.

Azure Sql Database [31] Az Azure Sql Database egy relációs adatbázis, ami lehetővé teszi Sql Server adatbázis tárolását a felhőben. Az erőforrások létrehozáskor konfigurálhatóak, de emellett automatikusan skálázódik, és 99.995%-os [24] elérést biztosít. A létrehozott adatbázishoz egy *ConnectionString*-el lehet csatlakozni, ami tartalmazza az adatbázis szerver minden információját.

NuGet Package Manager [34] A NuGet Package Manager csomagkezelő egy Microsoft által támogatott módszer arra, hogy fejlesztők különböző csomagokat, könyvtárakat osszanak meg egymás között. Definiálja a .Net-hez készült csomagok létrehozásának és felhasználásának módját és eszközöket nyújt ezekhez. Egy NuGet package kompilált kód és ehhez kapcsolódó fájlok gyűjteménye, amelyek információkat írnak le a csomagról. A fejlesztők létrehozhatnak csomagokat és megoszthatják őket publikus vagy privát platformon.

DinkToPdf [19] A DinkToPdf egy .NET csomag, amelyet a NuGet csomagkezelő segítségével lehet projektekhez hozzárendelni, és lehetővé teszi HTML oldalak PDF formátumba való konvertálását.

3.3. Web technológiák

React.js [48] A React.js a Meta Platforms által fejlesztett JavaScript könyvtár, amely felhasználói felületek fejlesztésére alkalmas. Alapja az újrahasznosítható komponensek használata, amelyeken belül a renderelési logika elve párosul más felhasználói felület logikával.

A React.js által biztosított JSX (JavaScript XML) szintaxis a komponensek HTML és JavaScript együttes használatával való leírását biztosítja. Ahhoz, hogy a böngésző képes legyen ezek megjelenítésére, az előfeldolgozók (mint például a Babel) a leírást JavaScript nyelvre fordítják. A komponensek megjelenítésének hátterét a Virtual DOM képezi, amely

[19] Ebben az esetben a secret egy titkos és egyedi karaktersort jelent, amelynek publikussá válása a hashelt információ felfedését okozza.

a felhasználói felület egy virtualizációja és amelyet a könyvtár az "eredeti" DOM-mal szinkronizál. Segítségével nem szükséges az oldalak teljes újratöltése, kizárólag azok a komponensek frissülnek, amelyek állapota megváltozott.

A 16.8-as verzióban bevezetett React Hooks API osztályok definiálása nélkül valósítja meg a komponensek állapot és életciklus jellemzőinek kezelését. Hook-oknak nevezett függvényeket biztosít, amelyek lehetővé teszik a függvénykomponensek használatát a fejlesztés során.

Bootstrap [4] A Bootstrap egy nyílt forráskódú CSS keretrendszer, amely reszponzív felhasználói felületek fejlesztésére szolgál. Emellett előre megírt, újrafelhasználható komponenseket is biztosít, amelyek lehetővé teszik a fejlesztők számára egy egységes kinézetű, praktikus platform létrehozását. A webalkalmazásban a 'bootstrap' és 'react-bootstrap' csomagok komponensei kerültek felhasználásra.

Node Package Manager (npm) [39] Az npm egy nyílt forráskódú csomagkezelő rendszer, amely node projektek struktúrált fejlesztésére szolgál. A projekt illetve a külső függőségek kezelése az általa nyújtott parancssori utasítások segítségével biztosított. A webalkalmazás projektje a 'create-react-app' npm parancs segítségével jött létre. A különböző előre definiált komponensek használata, illetve a nemzetköziesítés és routing megvalósítása érdekében külső npm csomagok felhasználása volt szükséges.

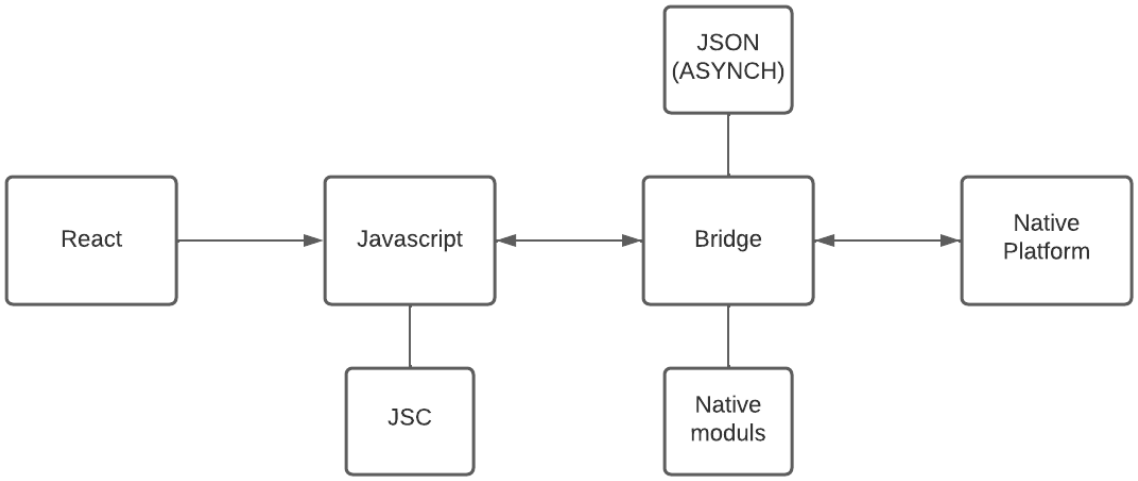
Pusher Channels API [26] A Pusher Channels egy hostolt API-szolgáltatás, amely lehetővé teszi valós idejű adatok web- és mobilalkalmazásokhoz való hozzáadását. Egy valós idejű kommunikációs réteget képez csatornák formájában, amelyeken keresztül a szerver rendszer-események formájában tud jelezni a kliensalkalmazásnak. A webalkalmazás információs oldala a szervertől kapott események feldolgozásával valósítja meg az eredmények valós idejű frissítését.

Heroku [55] A Heroku egy konténer-alapú PaaS[20], amely alkalmazások kitelepítésére és kezelésére nyújt lehetőséget. A platform biztosítja az infrastruktúra és a szoftver-összetevők karbantartását, így a fejlesztőknek csak a kódra kell összpontosítaniuk. A webalkalmazás kitelepítése a GitLab CI/CD-n belül definiált *pipeline* deploy *job* segítségével van megvalósítva, amely master ágra való *push* esetén a szolgáltatás heroku nevű távoli tárolóját szinkronizálja a projekt aktuális állapotával.

[20]Platform as a Service - Felhőalapú számítástechnikai modell, amely a felhőben hardver- és szoftvererőforrásokat biztosít a felhasználók számára az alkalmazások rugalmas fejlesztéséhez.

3.4. Mobil technológiák

React Native [49] A React Native a Meta Platforms, Inc. által fejlesztett, 2015-ben kiadott nyílt forráskódú JavaScript keretrendszer. Lehetőséget ad Android, iOS, macOs, Web és Windows platformokra egyszerre fejleszteni. Segítségével az alkalmazás használhatja a React keretrendszert, és a platform specifikus elemeit. Háttérben a React Native hidakat épít az applikáció és a célplatform között. Ezek a hidak felhasználják a React könyvtárat, és átadják a komponens hierarchiát a mobil eszközöknek, meghívják a natív megjelenítő API-kat, az Objective C-t felhasználva iOS esetén, a Java-t Android esetén. Például, React Native kódban egy nézet definiálható <*View*> tag-el, ami a háttérben Android-ra <*ViewGroup*>-ként, iOS-re <*UIView*>-ként, webes környezetben pedig <*div*>-ként fordítódik le.



11. ábra. Az ábra a React Native kapcsolatát mutatja be a natív platformokkal

Expo [16] Az Expo egy keretrendszer, és egy platform, amely segítségével React Native alkalmazások fejleszthetőek, építhetőek fel és telepíthetők ki. Az alkalmazást az Expo platform könnyen megoszthatóvá teszi más fejlesztőkel, ezzel gyorsítja a fejlesztési folyamatot. Két féle munkafolyamatra ad lehetőséget: menedzselt és tiszta. A menedzselt verzióban az Expo megpróbálja lekezelni az alkalmazás felépítésének a komplexitását, és csak egyetlen JavaScript/TypeScript kódbázis felel mindenért. Ez az egyszerűség viszont néha problémákat okozhat, ha valamelyik platformon teljes kontrollt szeretne a fejlesztő. Ilyenkor a tiszta munkafolyamat *expo prebuild* parancsával natív projekt generálható.

3.5. Eszközök

Git [56] A Git egy nyílt forráskódú osztott verziókövető rendszer. Követi a változásokat egy repository-n belül, így bármikor visszaállítható egy előző verzió, valamint könnyebbé teszi a közös munkát.

Gitlab [2] A Gitlab egy repository-menedzselő rendszer, mely tárolja a kódbázist, és segít követni verziókat, illetve a verziók közti különbségeket. Emellett lehetőséget nyújt a fejlesztési folyamat megtervezésére is, beépített Kanban[21] táblákkal rendelkezik.

Gitlab Continous Integration [20] A Gitlab CI/CD egy szoftverfejlesztési eszköz, mely egyesít három metodológiát: folyamatos integráció, kiadás és kitelepítés. Célja, hogy a hibákat már a fejlesztési folyamat során észre vegye, ezzel garantálva a kiadott alkalmazás minőségét.

Visual Studio Code [37] A Visual Studio Code egy forráskód-szerkesztő alkalmazás, mely számos nyelvet támogat, vizuális felületet szolgáltat a Git és az Azure termékek használatához, és formázza a forráskódot. Sok kiegészítő csomag áll rendelkezésre, melyek segítségével teljesen testre szabható az alkalmazás.

ESlint [18] Az ESLint egy statikus kódellenőrző, mely analizálja a kódot és megkeresi benne a formai és logikai hibákat, ezzel segít abban, hogy a kód minőségi maradjon a fejlesztés alatt.

Expo Go Az Expo Go az Expo mobilalkalmazása, melynek segítségével fizikai eszközökön tesztelhetjük a fejlesztés alatt álló alkalmazást. Gyorsítja a fejlesztési folyamatot, mivel a kódbeli változásokkal automatikusan frissül a telefonon az alkalmazás.

[21] Kanban - Lean munkamódszer a csoportos munka menedzselésére.

4. Az alkalmazás működése

4.1. Szerver beüzemelése

A szerver, ahogy a 2.1.2. alfejezetben is említve volt, csak a verseny napján van kitelepítve. A szerver kitelepítése az Azure-re parancssorból történik, a *master* branch-re érkező push műveleten keresztül. Ez a push elindít egy *pipeline*-t, ami létrehozza az erőforrásokat, úgy az adatbázisnak, mint az *Azure függvények* számára. Ezt a műveletet pipeline *job*-ok végzik el. Először létre lesznek hozva azok az összecsomagolt fájlok, amelyek a *függvényalkalamzásokhoz* fognak felkerülni. Ezután létrejönnek az adatbázis erőforrásai Azure oldalon, az erőforrás csoport, az adatbázis szerver és maga az adatbázis is. Ezt követően kerülnek bele az adatbázisba az entitásokhoz tartozó táblák, amelyek egy .NET osztály és egy SQL script segítségével állítódnak elő. Az adatbázis után létrejön az erőforráscsoport a függvényalkalmazások számára, ahová minden függvényalkalamazás kerül. Mindegyik függvényalkalmazás beállításai közé bekerül a *secret*, ami a JWT token generálásához használt kulcs, és a *ConnectionString*, ami az adatbázishoz való csatlakozást biztosítja.

Az entitásokhoz tartozó Azure függvények, a nekik megfelelő függvényalkalmazásba vannak kitelepítve, így nem kell egy entitáshoz köthető műveletek esetén az összes függvény aktiválódjon (cold start), hanem csak azok, amelyek az érintett entitáshoz tartoznak.

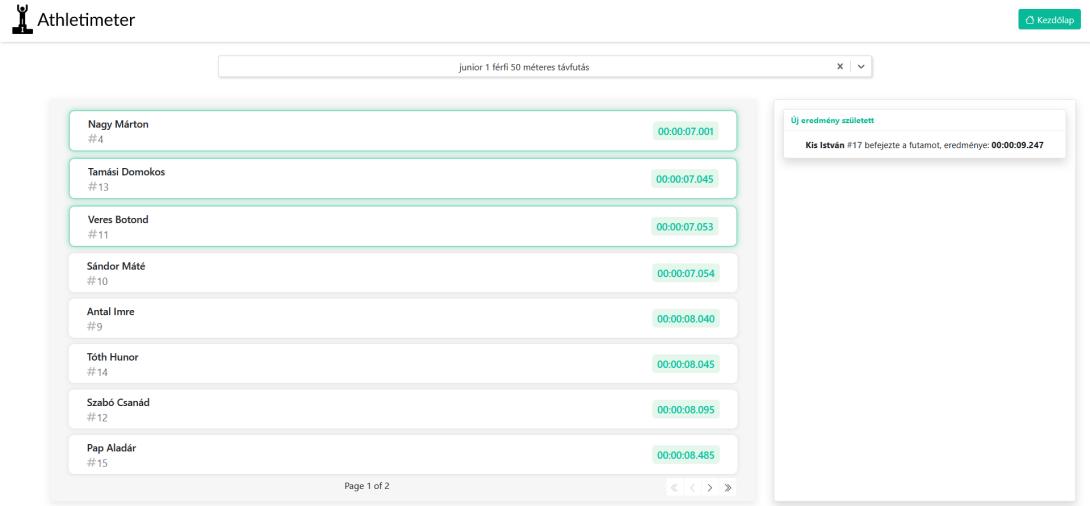
Az alkalmazás törlése ugyanúgy történik, mint a kitelepítése. A programozó a parancssorból hív egy push műveletet a *delete-resources* branchre, ahol ennek hatására lefut az Azure erőforrásokat letörlő pipeline. Ezt követően az Azure felületén nem marad semmilyen erőforrás.

4.2. Web kliens működése

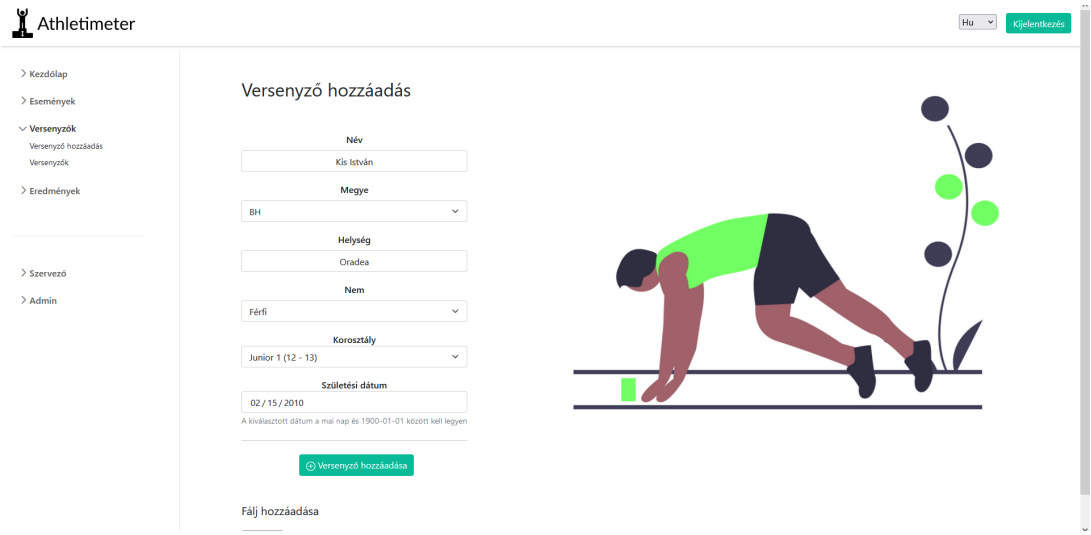
A webalkalmazás az 1.1. fejezetben meghatározott felhasználói csoportoknak nyújt szolgáltatásokat.

A verseny nézői vendégfelhasználóként tudják használni a platformot. Bejelentkezés nélkül lehetőségük van az oldal nyelvének megváltoztatására, illetve az információs oldal megtekintésére, amelyen valós időben nyomon követhetik a lenyíló listából kiválasztott versenyszám eredményeit. A funkcionalitást megvalósító oldalt a 12. ábra szemlélteti.

Bejelentkezés után az oldalmenün új szolgáltatások válnak elérhetővé. Mint ahogyan a 1.1.2. fejezetben is megfogalmazásra került, a szervezőknek a platform lehetőséget nyújt egy atlétikaversenyhez tartozó információk karbantartására. Az eltárolt adatok listázhatóak,



12. ábra. Az eredmények valós idejű követését lehetővé tevő információs oldal.

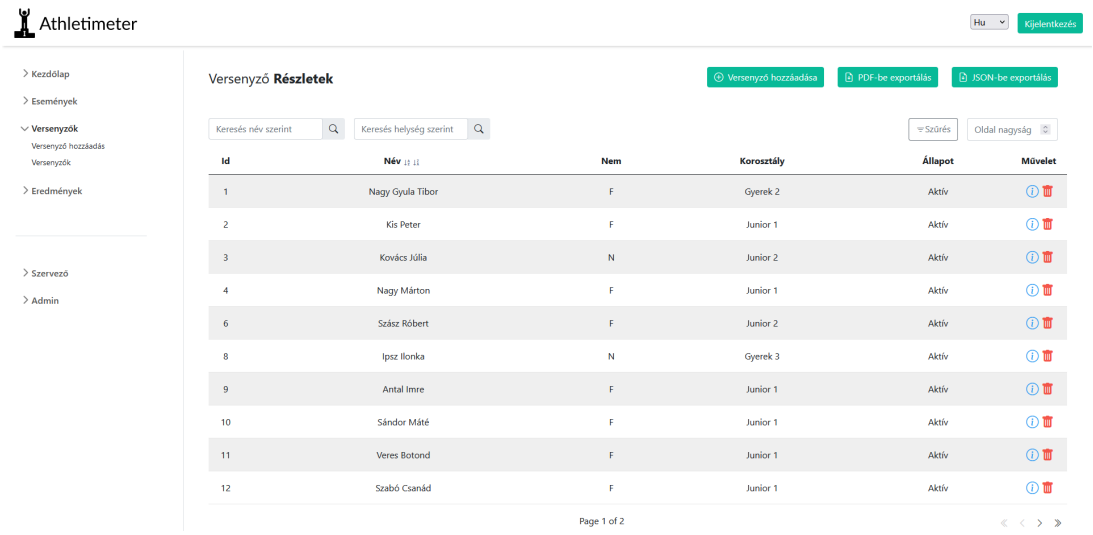


13. ábra. Versenyző hozzáadására szolgáló oldal.

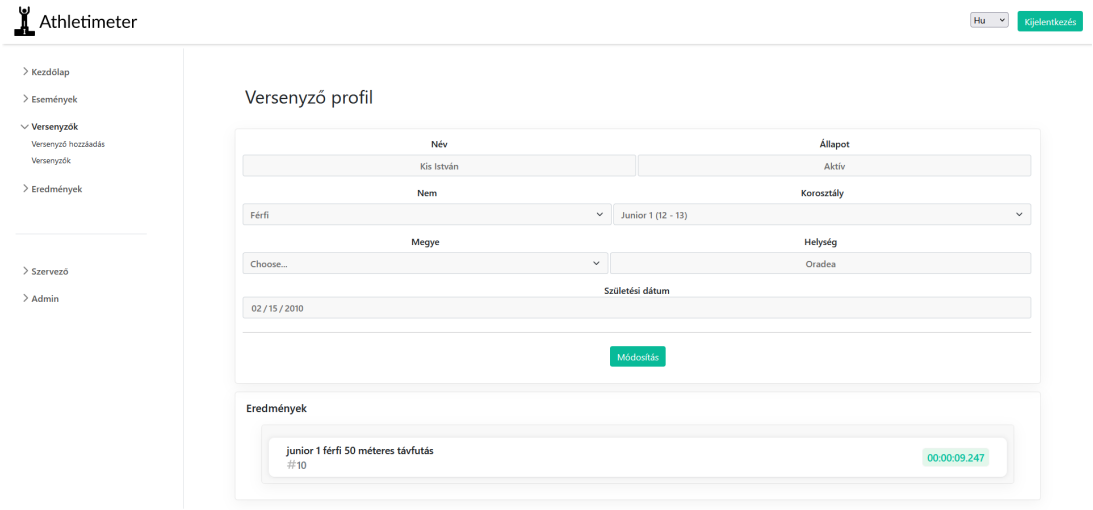
módosíthatóak, törölhetőek, kibővíthetőek új adatokkal, valamint a résztvevők, események és eredmények különböző formátumba archiválhatóak. A 13., 14., 15. ábrák a versenyzők kezelésére szolgáló oldalakat szemléltetik.

Abban az esetben, ha egy regisztrált versenyző, például sérülés miatt, nem tudja teljesíteni a futamokat, lehetőség van a státusz inaktívra állítására. Ilyenkor a teljesített versenyszámokon elért eredmények állapota is megfelelően változik. Ehhez hasonlóan lehet eljárni események és eredmények érvénytelenítése esetén is. Mivel egy atlétikaversenyt meghatározó adatokról van szó, ez az eljárás a törlési művelet egy biztonságos változatát jelenti. Korosztályok esetében a hagyományos törlési művelet alkalmazható.

Egy új eredmény felvezetése a 16. ábrán látható űrlap kitöltésével valósítható meg.



14. ábra. Versenyzők megtekintésére szolgáló oldal.

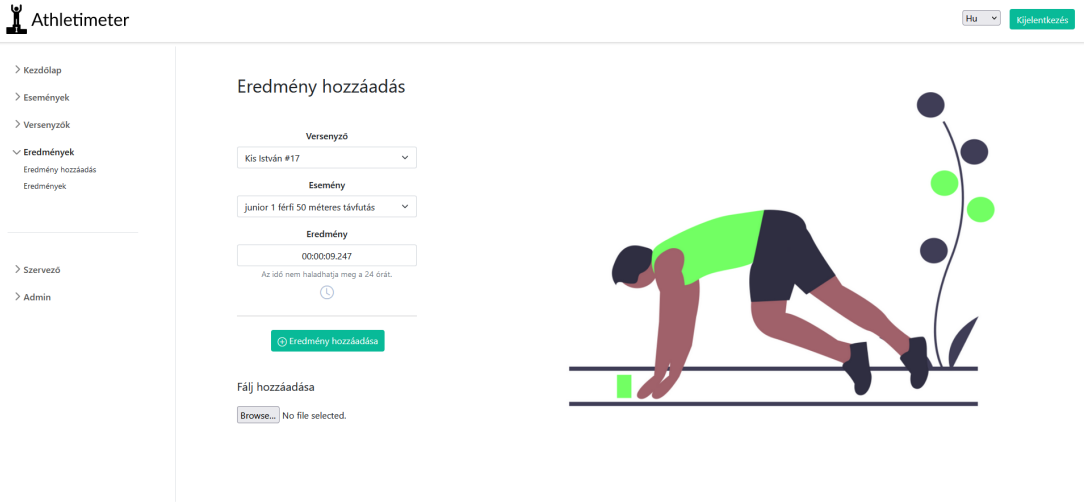


15. ábra. Versenyző adatainak módosítására szolgáló oldal.

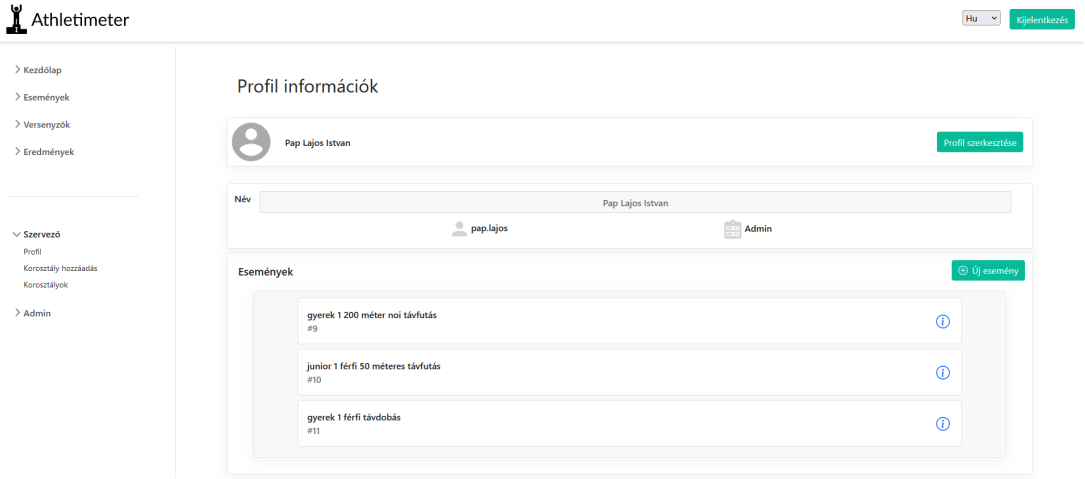
Amennyiben a lenyíló listákból kiválasztott versenyző és esemény beállított nem vagy korosztály attribútuma nem egyezik, egy megfelelő hibaüzenet jelzi az eltérést, és nem lehetséges a mentés elvégzése. Ellenkező esetben az eredmény az esemény mértékegységének megfelelő formátumban adható meg. Időben mérendő versenyszám esetén az óra ikonra kattintva, a képernyőn egy ablak ugrik fel, amely az időérték könnyebb és gyorsabb beírását teszi lehetővé.

Emellett a szervezők hozzáférnek a profiloldalhoz, amely a hozzájuk tartozó információk összesítésére szolgál. Megtekinthetik a felhasználónevüket, illetve az általuk létrehozott eseményeket. Rákattintva a Profil szerkesztése gombra, lehetőségük van az adminisztrátor által beállított név és jelszó megváltoztatására. Teljes név módosítása esetén, mentéskor az

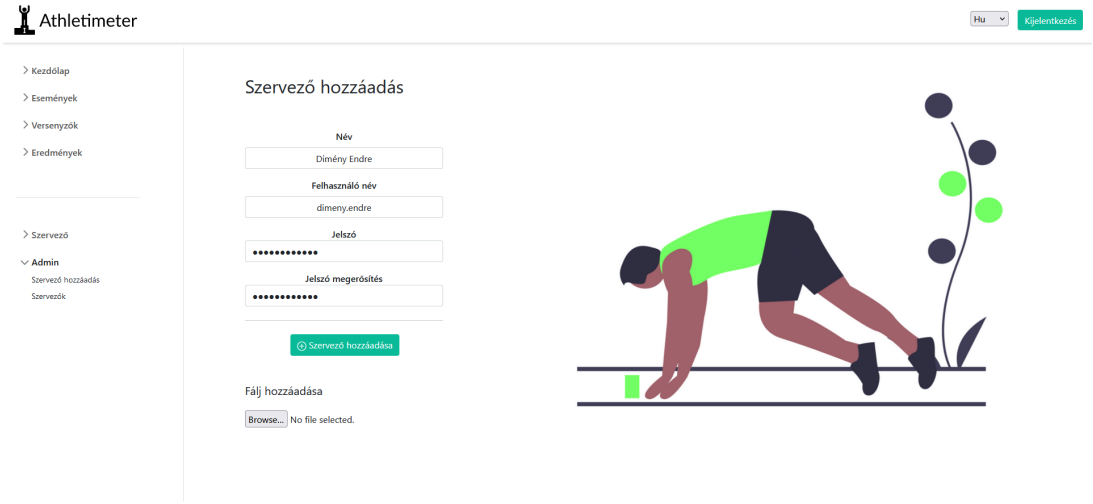
oldal megfelelően tükrözi az újonnan beállított nevet, anélkül, hogy szükséges lenne egy teljes oldal-újratöltés. Ezt a 17. ábra szemlélteti.



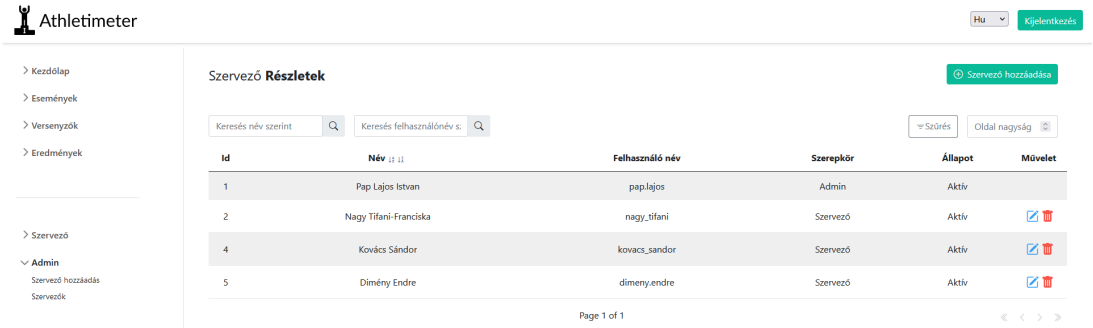
16. ábra. Eredmény hozzáadására szolgáló oldal.



17. ábra. Profil információk megtekintésére és módosítására szolgáló oldal.



18. ábra. Szervező hozzáadására szolgáló oldal.



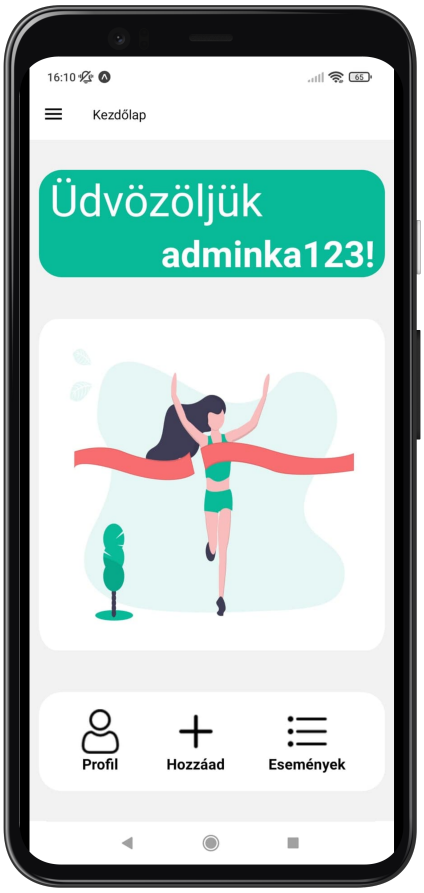
19. ábra. Szervezők megtekintésére szolgáló oldal.

Az adminisztrátor számára eggyel több funkcionalitás áll rendelkezésére, ugyanis a platformon lehetősége van a szervezők adatainak kezelésére. Ahhoz, hogy a verseny szervezői számára elérhetőek legyenek a weboldal szolgáltatásai, egy felhasználói fiókra van szükségük, amelyet az adminisztrátor egy név, egyedi felhasználónév és jelszó megadásával tud létrehozni. A szervezők hozzáadására és listázására szolgáló oldalakat a 18. és a 19. ábrák szemléltetik.

4.3. Mobil kliens működése

A mobil alkalmazás kizárólag a szervezőknek nyújt funkcionalitásokat, számukra könnyíti meg a verseny lebonyolítását a helyszínen, azzal, hogy többféleképpen vezethetnek fel eredményeket, áttekinthetik és kezelhetik a már eddig létező eredményeket, illetve listázhatják

20. ábra. A bejelentkező oldal



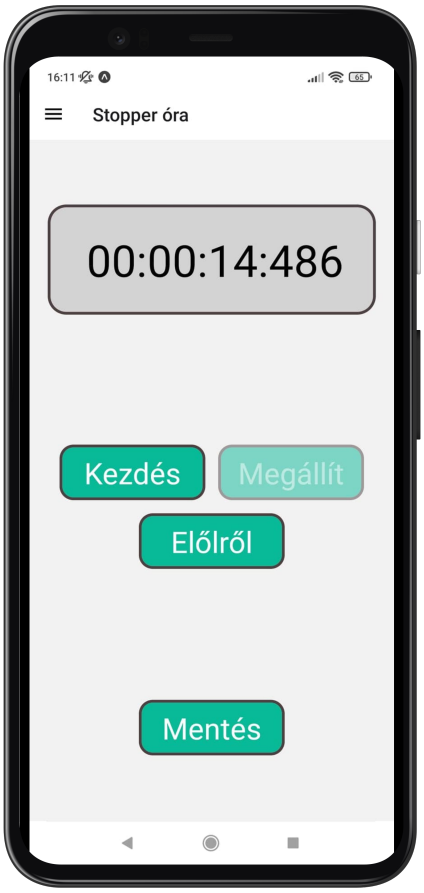
21. ábra. Az alkalmazás kezdőlapja

a versenyszámokat és ezek részleteit.

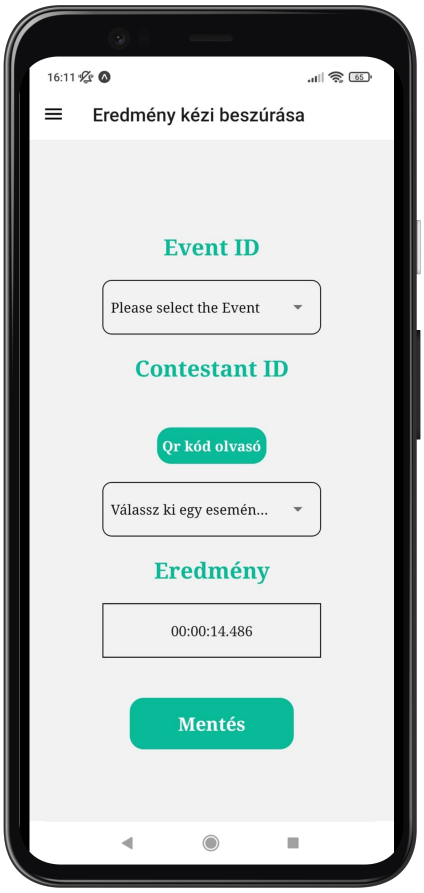
A mobil alkalmazás működése az Android platformon lesz bemutatva a következőkben, azzal a megjegyzéssel, hogy az alkalmazás néhány funkcionalitása jelenleg még nem elérhető iOS rendszeren, illetve a működő funkcionalitások esetében kis különbségek vannak a kinézetben, annak érdekében, hogy jobban beilleszkedjenek a platform stílusába.

Az alkalmazás elindításakor először kötelezően be kell jelentkezni szervezőként. Ugyanakkor kezdetben, majd bejelentkezés után a profil oldalon, lehetőség nyílik választani a magyar és az angol nyelv között, ahogy a 20. ábra szemlélteti.

Bejelentkezés után a 21. ábrán látható, hogy a főoldalról elérhetőek a Profil, Eredmény Hozzáadás és az Eseménylistázó oldalak, továbbá az oldalmenüt kihúzva ezek mellett elérhető az Eredmények listázó oldala, a Stopperóra, illetve a kijelentkezés opció.

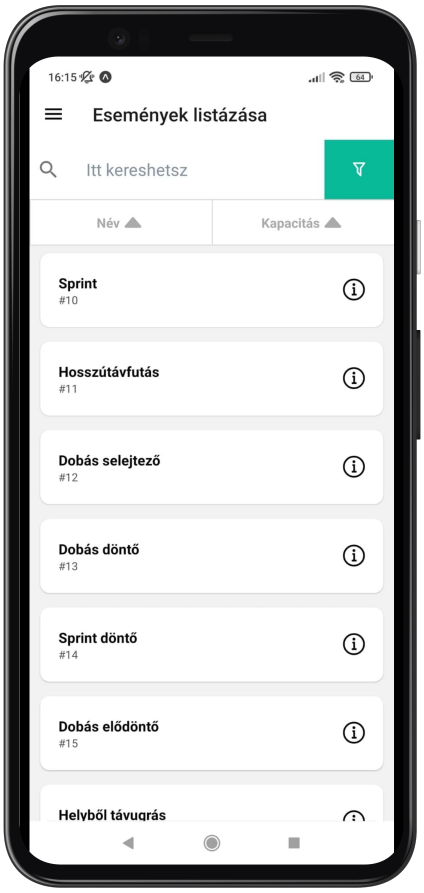


22. ábra. Stopperóra

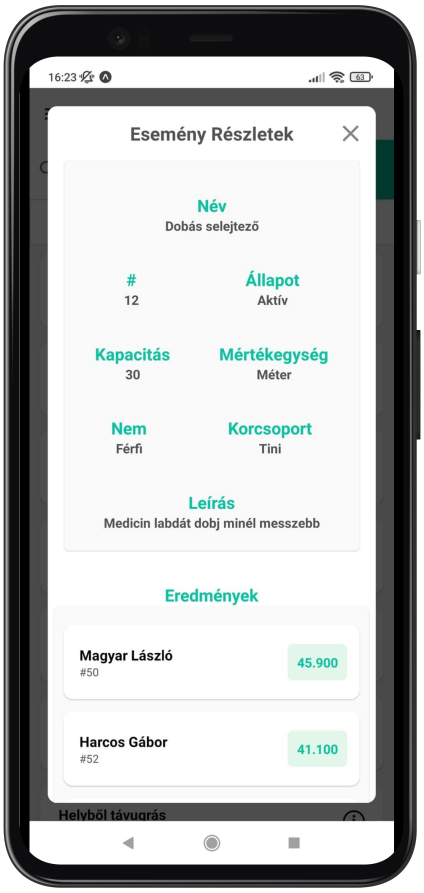


23. ábra. Eredmény beszúrása oldal

A 22. ábrán látható Stopperórát használva könnyebben rögzíthetőek az eredmények, ami után azonnal át lehet lépni az eredmény beszúró oldalra, amely a 23. ábrán van szemléltetve. A meglévő időérték mellett meg kell adni az adott esemény nevét, illetve azonosítani kell a versenyzőt. Ezt meg lehet tenni egy lenyíló listából, ahol megtalálható az összes aktív versenyző ID és névpárosa, vagy egy versenyzőt beazonosító QR kód beolvasásával is. Ilyenkor a QR olvasót a versenyző mezén levő QR kódra kell irányítani, mely az egyedi ID-ját tárolja. Csakis helyes, létező adatokat enged menteni az alkalmazás.



24. ábra. Eredmények listázása oldal



25. ábra. Eredmény részletei

Az esemény- és eredmény-listázó oldalak a 24. és 25. ábrákon vannak szemléltetve, ezeken megtekinthetők, rendezhetők és szűrhetők az adatok. Hosszan nyomva egy sort megtekinthető annak a részletes leírása, valamint eredmények esetében törölhetők és szerkeszthetők is a már meglévő eredmények.

Következtetések és továbbfejlesztési lehetőségek

Az Athletimeter alkalmazás fejlesztése során egy olyan három komponensből álló szoftverrendszer született, amely az adatok digitális feldolgozásával egy atlétikaverseny lebonyolítását hivatott megkönnyíteni.

Mivel az alkalmazás atlétikaversenyek során kerül felhasználásra, nem szükséges ennek állandó elérhetősége. Az előre megírt script-ek segítségével lehetőség van az alkalmazásszerver verseny előtti automatikus beüzemelésére, és a futamok lezajlása után ennek gyors eltávolítására.

A rendszer webalkalmazása egy átlátható és könnyen kezelhető felületet biztosít a szervezők számára egy verseny adatainak karbantartására. A szervezők mellett, a platform a verseny nézőinek is biztosítja az eredmények valós idejű követésének lehetőségét, hozzájárulva a szurkolói élmény növeléséhez.

A mobil applikáción keresztül a szervezők olyan szolgáltatásokat érnek el, amelyek a versenyzők beazonosítását, illetve az általuk elért eredmények mérését/mentését valósítják meg.

A fejlesztőcsapat, illetve a mentorok és a vezetőtanár részéről több továbbfejlesztési lehetőség is megfogalmazódott.

A szerveroldalon a meglévő scripteket ki lehetne bővíteni, hogy személyre szabhatók legyenek a felhőben tárolt erőforrások, így egyszerre több verseny lebonyolítása is megvalósítható lenne.

Jelenleg a webalkalmazás használatával az adatok különböző formátumokba archiválhatóak, viszont ezek statisztikai feldolgozása nem biztosított. A szervezők számára lehetőséget lehetne adni arra, hogy a webes platform egy oldalán megtekinthessék az eltárolt eredmények alapján a különböző kritériumok szerint kigenerált statisztikákat és diagrammokat.

Az információs oldal többféle nézettel való bővítése a beérkező eredmények különböző típusú megjelenítésére szolgálna. A verseny nézői, a lista formátum mellett, az adatok másféle vizualizációját is választhatnák, amely élvezhetőbbé tenné ezek nyomon követését.

A mobilalkalmazás bizonyos funkciói jelenleg csak Androidon érhetők el, valamit a kitelepítés is csak ezen a platformon valósul meg. Így az iOS platformra adaptálás is egy továbbfejlesztési lehetőség.

Mivel a mezre ragasztott QR kód esetében a futamok során problémák merülhetnek fel, a versenyzők beazonosításának egy alternatívájaként NFC[22] olvasót lehetne használni. A

[22]Near Field Communiation - Rövid hatótávú kommunikációs technológia, amelyet eszközök közötti

résztvevők a futamok során egy NFC karkötővel rendelkeznének, amelyet a szervezők a mobil applikáció segítségével tudnának beolvasni. Ez lehetőséget adna időben mérendő események esetében az eredmények beszúrásának automatizálására is, ugyanis a célnál szenzorokkal ellátott NFC kapuk lehetnének elhelyezve, amelyek érzékelnék a versenyző beérkezését.

adatcserére alkalmaznak.

Hivatkozások

[1] The Kubernetes Authors. *What is Kubernetes?* 2022. URL: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (elérés dátuma: 2022. ápr. 14.)

[2] GitLab B.V. *GitLab*. 2022. URL: https://about.gitlab.com/stages-devops-lifecycle/ (elérés dátuma: 2022. ápr. 18.)

[3] Babel. *What is Babel?* 2015. URL: https://babeljs.io/docs/en/index.html (elérés dátuma: 2022. ápr. 11.)

[4] Bootstrap. *Bootstrap*. URL: https://getbootstrap.com/ (elérés dátuma: 2022. ápr. 15.)

[5] IBM Corporation. *Rich client platform architecture*. 2021. URL: https://www.ibm.com/docs/en/order-management-sw/9.2.1?topic=reference-rich-client-platform-architecture (elérés dátuma: 2022. ápr. 11.)

[6] Mozilla Corporation. *AbortController*. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/API/AbortController (elérés dátuma: 2022. ápr. 14.)

[7] Mozilla Corporation. *Cookies doucmentation*. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie (elérés dátuma: 2022. ápr. 18.)

[8] Mozilla Corporation. *Element.getBoundingClientRect()*. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/API/Element/getBoundingClientRect (elérés dátuma: 2022. ápr. 19.)

[9] Mozilla Corporation. *Fetch API*. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (elérés dátuma: 2022. ápr. 5.)

[10] Mozilla Corporation. *MVC*. 2022. URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC (elérés dátuma: 2022. ápr. 11.)

[11] Mozilla Corporation. *Promise*. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise (elérés dátuma: 2022. ápr. 8.)

[12] Mozilla Corporation. *SPA*. 2021. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (elérés dátuma: 2022. ápr. 8.)

[13] Mozilla Corporation. *Window.requestAnimationFrame()*. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame (elérés dátuma: 2022. ápr. 19.)

[14] Rick Anderson Daniel Roth és Shaun Luttin. *Introduction to ASP.NET Core*. 2022. URL: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0 (elérés dátuma: 2022. ápr. 17.)

[15] Seres János Dr. Béres Sándor Dr. habil Müller Anetta és Szalay Gábor. *Atlétika*. EKF Líceum Kiadó, 2015.

[16] Expo. *Expo*. 2022. URL: https://docs.expo.dev/ (elérés dátuma: 2022. ápr. 13.)

[17] Expo. *Expo SecureStore doucmentation*. 2022. URL: https://docs.expo.dev/versions/latest/sdk/securestore/ (elérés dátuma: 2022. ápr. 18.)

[18] Open JS Foundation. *ESLint*. 2022. URL: https://eslint.org/ (elérés dátuma: 2022. ápr. 18.)

[19] Rok Dvojmoč - Github. *DinkToPdf documentation*. 2022. URL: https://github.com/rdvojmoc/DinkToPdf (elérés dátuma: 2022. ápr. 17.)

[20] Gitlab. *Gitlab CI/CD*. 2022. URL: https://docs.gitlab.com/ee/ci/ (elérés dátuma: 2022. ápr. 18.)

[21] Auth0® Inc. *JWT Token*. 2022. URL: https://jwt.io/introduction (elérés dátuma: 2022. ápr. 11.)

[22] IntelliPaat. *AWS vs Azure vs Google Cloud*. 2021. URL: https://intellipaat.com/blog/aws-vs-azure-vs-google-cloud/#no1 (elérés dátuma: 2022. ápr. 20.)

[23] IntelliPaat. *AWS vs Azure vs Google Cloud: Pricing*. 2021. URL: https://intellipaat.com/blog/aws-vs-azure-vs-google-cloud/#no8 (elérés dátuma: 2022. ápr. 20.)

[24] IntelliPaat. *Azure Sql database Service Level Agreement*. 2021. URL: https://intellipaat.com/blog/aws-vs-azure-vs-google-cloud/#no8 (elérés dátuma: 2022. ápr. 20.)

[25] Paul Lewis. *FLIP Your Animations*. 2015. URL: https://aerotwist.com/blog/flip-your-animations/ (elérés dátuma: 2022. ápr. 14.)

[26] Pusher Ltd. *Channels overview*. 2020. URL: https://pusher.com/docs/channels/ (elérés dátuma: 2022. ápr. 15.)

[27] Pusher Ltd. *Client API overview*. 2020. URL: https://pusher.com/docs/channels/using_channels/client-api-overview/ (elérés dátuma: 2022. ápr. 7.)

[28] Pusher Ltd. *Events*. 2020. URL: https://pusher.com/docs/channels/using_channels/events/ (elérés dátuma: 2022. ápr. 14.)

[29] Microsoft. *Azure Functions*. 2022. URL: https://docs.microsoft.com/en-us/azure/azure-functions/ (elérés dátuma: 2022. ápr. 11.)

[30] Microsoft. *Azure Functions Core Tools*. 2022. URL: https://docs.microsoft.com/en-us/azure/azure-functions/functions-run-local?tabs=v4 (elérés dátuma: 2022. ápr. 17.)

[31] Microsoft. *Azure SQL Database*. 2022. URL: https://docs.microsoft.com/en-us/azure/azure-sql/ (elérés dátuma: 2022. ápr. 21.)

[32] Microsoft. *Entity Framework documentation*. 2022. URL: https://docs.microsoft.com/en-us/ef/ (elérés dátuma: 2022. ápr. 17.)

[33] Microsoft. *Entity Framework documentation*. 2022. URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/ (elérés dátuma: 2022. ápr. 17.)

[34] Microsoft. *Entity Framework documentation*. 2022. URL: https://docs.microsoft.com/en-us/nuget/ (elérés dátuma: 2022. ápr. 17.)

[35] Microsoft. *Introduction to .NET Core*. 2022. URL: https://docs.microsoft.com/en-us/dotnet/core/introduction (elérés dátuma: 2022. ápr. 17.)

[36] Microsoft. *Microsoft Azure*. 2022. URL: https://azure.microsoft.com/en-us/ (elérés dátuma: 2022. ápr. 11.)

[37] Microsoft. *Visual Studio Code*. 2022. URL: https://code.visualstudio.com/docs (elérés dátuma: 2022. ápr. 18.)

[38] React Navigation. *React Navigation*. 2022. URL: https://reactnavigation.org/docs/getting-started (elérés dátuma: 2022. ápr. 13.)

[39] *npm Docs*. URL: https://docs.npmjs.com/ (elérés dátuma: 2022. ápr. 15.)

[40] Ravindu Shehan Perera. *Pure and Impure Functions in JavaScript: A Complete Guide*. 2022. URL: https://www.syncfusion.com/blogs/post/pure-and-impure-functions-in-javascript-a-complete-guide.aspx (elérés dátuma: 2022. ápr. 12.)

[41] Meta Platforms. *componentDidMount*. 2022. URL: https://reactjs.org/docs/react-component.html#componentdidmount (elérés dátuma: 2022. ápr. 12.)

[42] Meta Platforms. *componentDidUpdate*. 2022. URL: https://reactjs.org/docs/react-component.html#componentdidupdate (elérés dátuma: 2022. ápr. 12.)

[43] Meta Platforms. *Components and Props*. 2022. URL: https://reactjs.org/docs/components-and-props.html (elérés dátuma: 2022. ápr. 11.)

[44] Meta Platforms. *componentWillUnmount*. 2022. URL: https://reactjs.org/docs/react-component.html#componentwillunmount (elérés dátuma: 2022. ápr. 12.)

[45] Meta Platforms. *Context*. 2022. URL: https://hu.reactjs.org/docs/context.html (elérés dátuma: 2022. ápr. 14.)

[46] Meta Platforms. *createElement()*. 2022. URL: https://reactjs.org/docs/react-api.html#createelement (elérés dátuma: 2022. ápr. 11.)

[47] Meta Platforms. *Hooks API Reference*. 2022. URL: https://reactjs.org/docs/hooks-reference.html (elérés dátuma: 2022. ápr. 12.)

[48] Meta Platforms. *React*. 2022. URL: https://reactjs.org/ (elérés dátuma: 2022. ápr. 15.)

[49] Meta Platforms. *React Native*. 2022. URL: https://reactnative.dev (elérés dátuma: 2022. ápr. 9.)

[50] Meta Platforms. *useEffect*. 2022. URL: https://reactjs.org/docs/hooks-reference.html#useeffect (elérés dátuma: 2022. ápr. 12.)

[51] Meta Platforms. *useLayoutEffect*. 2022. URL: https://reactjs.org/docs/hooks-reference.html#uselayouteffect (elérés dátuma: 2022. ápr. 19.)

[52] Meta Platforms. *useReducer*. 2022. URL: https://reactjs.org/docs/hooks-reference.html#usereducer (elérés dátuma: 2022. ápr. 12.)

[53] Meta Platforms. *useState*. 2022. URL: https://reactjs.org/docs/hooks-reference.html#usestate (elérés dátuma: 2022. ápr. 12.)

[54] Leonard Richardson és Mike Amundsen. *RESTful Web APIs*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2013.

[55] Salesforce.com. *What is Heroku?* 2022. URL: https://www.heroku.com/about (elérés dátuma: 2022. ápr. 17.)

[56] Jason Long Scott Chacon. *Git*. 2022. URL: https://git-scm.com/ (elérés dátuma: 2022. ápr. 18.)

[57] Simplilearn Solutions. *What is Azure?* 2022. URL: https://www.simplilearn.com/tutorials/azure-tutorial/what-is-azure?source=sl_frs_nav_playlist_video_clicked#what_is_microsoft_azure (elérés dátuma: 2022. ápr. 20.)