XXIII. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)
Kolozsvár, 2020. május 21–24.

DiáknApp

Csapatok közti összetett bajnokságok szervezésére és lebonyolítására alkalmas szoftverrendszer

Szerzők:

Erdőközi Virág
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év
László Norbert
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év
Mekker Krisztián
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:

dr. Sulyok Csaba, tanársegéd
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar
Ráduly Sándor, szoftverfejlesztő,
Codespring
Gagyi Mátyás, szoftverfejlesztő,
Codespring

CODESPRING
OUR CODE YOUR SUCCESS

Kivonat

A diáknapok az év egyik legnagyobb rendezvényének számít az egyetemisták körében, ahol minden májusban közel 2000 fiatal mérkőzik meg különböző versenyszámokban. A diáknapok programjainak szervezése és kivitelezése különféle adatok bevezetésével, nyilvántartásával és kezelésével jár, úgy a szervezők, mint a csapatok részéről.

A DiáknApp egy olyan felületet biztosít a diákszövetségek számára, amely megkönnyíti az adatok rendszerbe való bevitelét és számontartását, valamint a programok áttekinthetőségét: lehetővé teszi a különböző események, illetve az ezek keretein belül zajló versenyek naptárba való rögzítését, a csapatok ezekre való benevezését, valamint a versenyek alakulásának nyomonkövetését.

A rendszer egy saját központi API szerverrel, valamint három klienssel rendelkezik: egy Android és iOS alkamazás, illetve egy webes felület. Mindhárom klienst a szerver szolgálja ki, a felhasználók pedig bármelyik felületen bevihetnek, vagy lekérhetnek bizonyos adatokat.

A dolgozat a DiáknApp projekt részletes bemutatására vállalkozik, kitérve az architektúra, illetve a fő komponensek megvalósítására, és az ezekkel kapcsolatos főbb döntések megalapozottságára, a felhasznált technológiákra, valamint az igénybe vett módszerekre és eszközökre.

Tartalomjegyzék

Bevezető

A diáknapok az egyetemista élet egyik kihagyhatatlan rendezvénysorozata, ahol minden májusban a fiatalok mérkőzhetnek meg különböző versenyszámokban, jól érezhetik magukat, akár versenyzőként, akár nézőként is. A diáknapok eseményei nem csak csapatok közötti versengésből áll, hanem fellépések, bulik és más szociális programok is megszervezésre kerülnek, amelyekhez bárki csatlakozhat. A diákok értesülhetnek a közelgő eseményekről vagy akár megtekinthetik a más városokban szervezett diáknapok programját is.

A diákok csapatokat alkotnak, amelyben a barátaikkal együtt részt vesznek a diáknapok eseményein. A 2019-es kolozsvári diáknapok keretein belül 55, 35-38 főből álló csapat jelentkezett, amely körülbelül 2000 fiatalt jelent. A Kolozsvári Magyar Diákszövetség (KMDSz) szerint tavaly körülbelül ötezer egyetemistát [21] mozgatott meg az egész rendezvénysorozat. Nem csak Kolozsváron, hanem más városokban is szerveznek diáknapokat: Marosvásárhelyi diáknapok a Marosvásárhelyi Magyar Diákszövetség szervezésében; Háromszéki Diáknapok a Kovászna megyei diákok számára és még sok hasonló esemény különböző városokban.

A diáknapok szervezőinek nagy kihívás egy ilyen nagy színvonalú esemény lebonyolítása, a sok program és verseny megszervezése mellett számon kell tartaniuk a csapatokat és azok kapitányait, melyek azok a csapatok amelyek regisztráltak, melyek késték le a határidőt, ellenőrizni a diákok adatait és még sok más apró fontos részletre is oda kell figyelniük. Így ez a folyamat hosszas és időigényes, amelyben gyakran felmerülhetnek félreértések, esetleges rossz adatok bevitele.

A DiáknApp projekt célja pontosan ennek a folyamatnak a megkönnyítése, azaz, hogy segítsen a diákszövetségeknek a sok adatot számontartani, valamint hogy egy egységes felületet biztosítson a diáknapokhoz hasonló események megszervezésére.

A platform lehetővé teszi a szervezők számára, hogy létrehozzanak úgynevezett bajnokságokat, illetve az ezek keretein belül zajló versenyek naptárba való rögzítését. A csapatkapitányok könnyen tudnak létrehozni csapatokat, amelyekkel regisztrálnak az adott diáknapokra. Ezeket a kéréseket pedig a szervezők elfogadhatják, vagy elutasíthatják. Miután megkezdődött a diáknapok a szervezők pontozhatják a csapatok teljesítményét a különböző versenyeken, amelyeket a csapatok vagy más érdekeltek is megtekinthetnek.

Az alkalmazás biztosít egy webes felületet, valamint egy Android és egy iOS natív mobilalkalmazást is, ahol megtekinthetőek a bajnokságok, események és csapatok. Azonkívül a szervezőknek segítséget nyújt a bajnokságok nyilvántartásában, a felhasználóknak pedig megadja a lehetőséget, hogy könnyedén jelentkezhessenek csapattársaikkal a bajnokságokra.

A dolgozat a következőképpen van struktúrálva: az 1. fejezetben az alkalmazásban

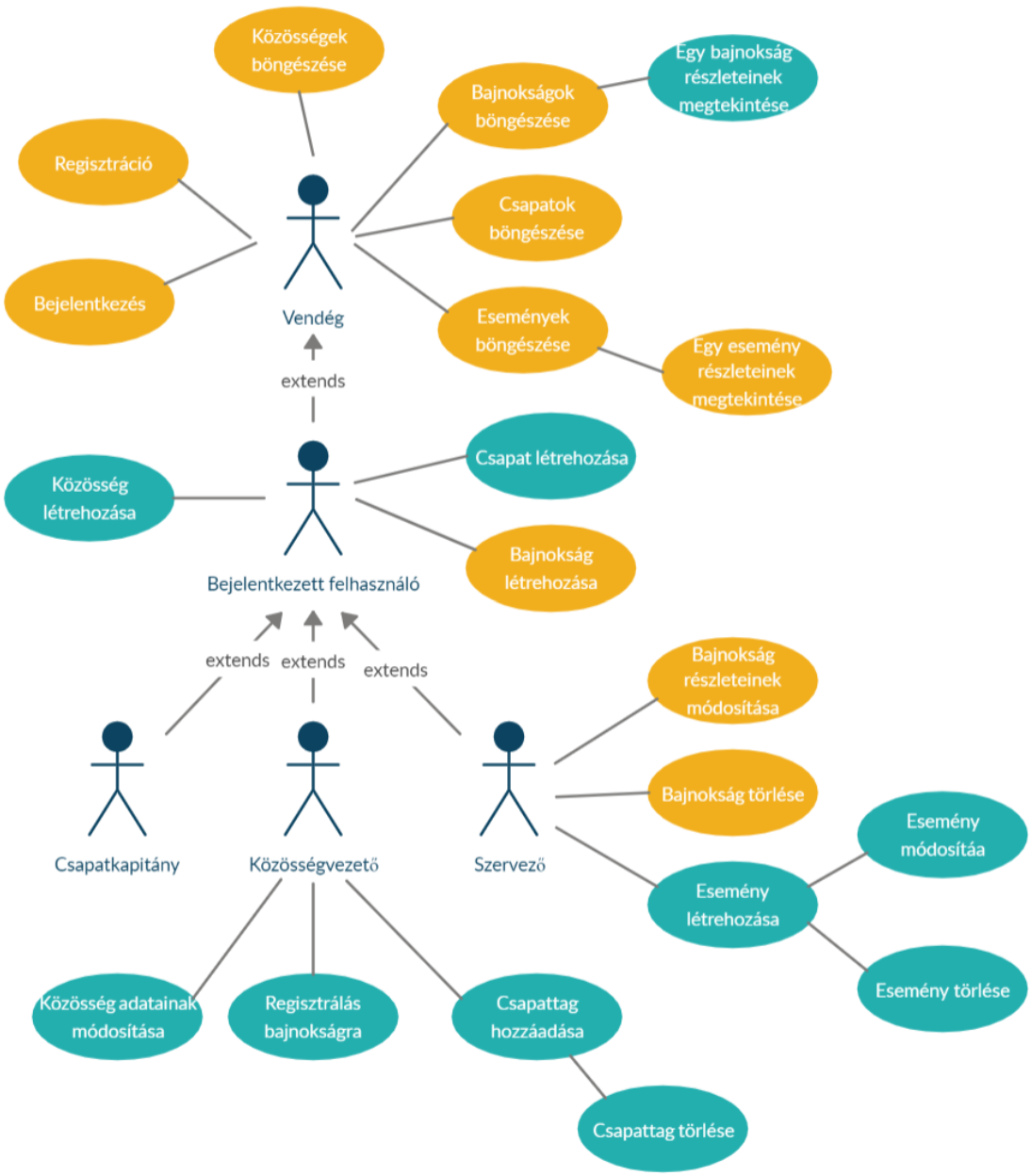
megvalósított funkcionalitások, valamint az egyes felhasználók jogosultságai vannak bemutatva. A 2. fejezet részletesen bemutatja az alkalmazás működését, külön a szervert és a három kliensalkalmazást. A 3. fejezet leírja a felhasznált technológiákat és eszközöket, valamint a 4. fejezet bemutatja az alkalmazás működését.

A projekt fejlesztése 2019 nyarán kezdődött a Codespring Mentorprogram szakmai gyakorlatának részeként. A 2019-2020-as tanévben a *Csoportos projekt* tantárgy keretén belül egy félévig csatlakozott hozzánk három évfolyamtársunk is: Botos Barbara, Lukács Levente-Ferencz és Lőrincz Péter.

Ezúton szeretnénk megköszönni a diáktársaink hozzájárulását, mentorainknak Sulyok Csabának, Gagyi Mátyásnak és Ráduly Sándornak a szakmai segítséget és támogatást.

1. Funkcionalitások

A DiáknApp projekt célja az, hogy egy olyan platformot biztosítson, ahol könnyedén átláthatóak a különféle bajnokságok, valamint azok programjai és a csapatok, amelyek részt vesznek a mérkőzéseken. A csapatok ezen a platformon könnyedén tudnak regisztrálni a bajnokságokra. A bajnokságok szervezőinek pedig lehetőséget ad arra, hogy az eseményeket létrehozzák, vagy módosítsák azokat, valamint hogy a csapatok regisztrációit láthassák és ezeket elfogadják vagy elutasítsák.



1. ábra. A felhasználók közötti öröklődés és funkcionalitások. A kékkel jelölt használati esetek a webes platform funkcionalitásait jelölik, a narancssárgák pedig a mobil alkalmazásokban is megtalálható funkcionalitásokat.

Ezen funkcionalitások nem elérhetőek akárki számára, ezért egy felhasználónak különféle szerepkörei lehetnek (lásd 1. ábra), amelyek eltérő jogosultságokkal rendelkeznek. A felhasználókat elsősorban két fő kategóriába lehet sorolni, mégpedig: *vendég* (nem bejelentkezett) felhasználó, illetve *bejelentkezett* felhasználó. A legkevesebb jogosultsága a vendég felhasználónak van, az adatokat csak megtekintheti, de nem módosíthatja.

Ezen kívül a bejelentkezett felhasználóknak háromféle szerepköre lehet: *csapatkapitány*, *közösségvezető* és *szervező*. Egy felhasználóhoz nem csak egy szerepkör tartozhat, hanem attól függően, hogy milyen feladatkört lát el egy diáknapokon belül, többféle szerepköre is lehet. Közösségvezetői szerepkörrel rendelkeznek azok a felhasználók, akik létre hoztak egy adott közösséget. Ezek a felhasználók tudnak regisztrálni csapatot a bajnokságokra, ahol később *csapatkapitány* szerepkört töltenek be. Csapatkapitány lehet egy egyszerű bejelentkezett felhasználó is, mivel lehetőség adódik, hogy csapattársak megadásával jelentkezzenek egy bajnokságra. Továbbá az a felhasználó, aki létrehozott egy adott bajnokságot *szervezői* szerepkörrel rendelkezik.

A dolgozat ezen fejezete részletesen tárgyalja a felhasználók jogosultágait, szerepkörökre lebontva.

1.1. Vendég felhasználó jogosultságai

Az alkalmazás használatához nem szükséges bejelentkezni, viszont ilyenkor csak bizonyos funkcionalitások lesznek elérhetőek. Megtekinthető a diáknapok listája, az egyiket kiválasztva pedig további részletek jelennek meg a bajnokságról, valamint a hozzá tartozó események, illetve a csapatok ranglistája is láthatóvá válik. Ezen kívül a közösségek is böngészhetőek, ezekből egyiket kiválasztva pedig részletesebb információk jelennek meg a közösségről. Egy vendég felhasználónak lehetősége van regisztrálni és bejelentkezni, amelyet megtehet az adatainak a megadásával, vagy akár a Facebook fiókjával is.

1.2. Bejelentkezett felhasználó jogosultságai

Egy bejelentkezett felhasználó rendelkezik az 1.1. részben bemutatott funkcionalitásokkal, vagyis rendelkezik a nem bejelentkezett felhasználó jogaival is. Minden bejelentkezett felhasználó - szerepkörtől függetlenül - létrehozhat egy új közösséget, amelynek ő lesz a vezetője. Így a felhasználónak a szerepköre megváltozik *közösségvezetőre* és rendelkezni fog azon szerepkör jogaival is (lásd 1.2. fejezet). Ezen kívül egy felhasználó regisztrálni tud egy új csapatot a bajnokságokra, ahol csapatkapitány szerepkörrel fog rendelkezni (lásd 1.2. fejezet), valamint szintén joga van létrehozni egy új bajnokságot, ahol a felhasználó *szervezői* szerepkört kap (lásd 1.2. fejezet).

Csapatkapitány szerepkör

Egy csapatkapitány szerepkörrel rendelkező felhasználó rendelkezik az eddig bemutatott funkcionalitásokkal. Külön jogosultságokkal nem rendelkezik, a szerepkör jelenléte azért fontos, hogy a szervezők tudják melyik felhasználó egy adott csapat kapitánya.

Közösségvezetői szerepkör

Közösségvezetői szerepkörrel rendelkeznek azok a felhasználók, akik létrehozták az adott közösséget. A közösségek vezetői jelentkezhetnek különböző bajnokságokra a közösség tagjainak kiválasztásával, így csapatkapitány jogosultságot kapnak. A közösség és a csapat között az a különbség, hogy egy csapat csak egy adott bajnokságon belül létezik, amíg a közösség nem függ egy bajnokságtól sem. A közösségbe tartozó emberek lehetnek csapattagok, szimpatizánsok, esetleg régebbi csapattagok is.

A közösség vezetőjének a szerepkörébe tartozik az, hogy módosíthatja az információkat az adott közösségről, hozzáadhat új tagokat vagy esetleg eltávolíthatja azokat, akik szerinte nem kellene tagjai legyenek a közösségnek.

Szervezői szerepkör

Szervezői szerepkörrel rendelkeznek azok a felhasználók, akik a rendszeren belül létrehoztak egy bajnokságot. Így a szerepkör egyik legfontosabb jogosultága, hogy módosítani tud információkat a bajnoksággal kapcsolatban, esetleg törölni is tudja azt, minden hozzá tartozó eseménnyel és csapattal együtt. Egy másik jogosultsága, hogy létre tud hozni eseményeket, módosítani tudja azokat, vagy esetleg törölni, hogyha nem találja megfelelőnek azokat. Azon kívül moderálni tudja a csapatok jelentkezését is az adott bajnokságon belül, meg tudja tekinteni a csapatok listáját és ennek függvényében eldöntheti, hogy engedélyezi a csapat részvételét az adott diáknapokon, vagy elutasítja azt.

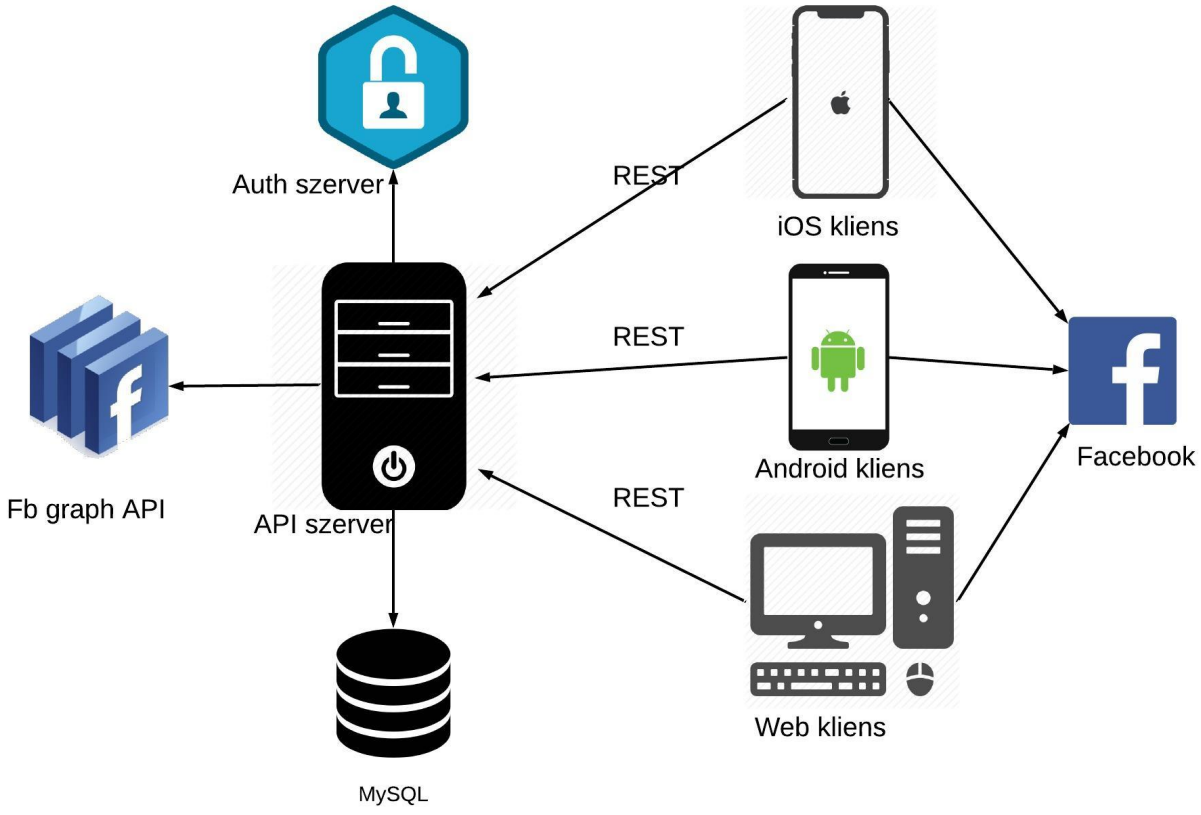
2. Az alkalmazás felépítése

A DiáknApp szoftverrendszer biztosít egy webes platformot, illetve natív Android és iOS mobilalkalmazásokat is. Ezenkívül biztosít egy szerveralkalmazást is, amellyel a kliensalkalmazások REST kéréseken keresztül tudnak kommunikálni. Ahogyan a 2. ábra is mutatja, a szerver egy MySQL relációs adatbázisban tárolja az adatokat, valamint a felhasználók biztonságos bejelentkezéséért egy OAuth providerként működő hitelesítési szerver felelős.

A bejelentkezés kétféleképpen történhet: standard módon, illetve Facebook fiók megadásával. Standard bejelentkezés esetén a felhasználó megadja az adatait, amit a hitelesítési szerver ellenőriz. A Facebook bejelentkezéshez a kliensalkalmazások a Facebook API [19] használatával kérik le a felhasználó adatait, amelyek a szerveroldalon is validálva vannak a Facebook Graph Api által.

2.1. Az alkalmazásszerver

A DiáknApp szoftverrendszer az adatok központi tárolása érdekében biztosít egy szerveralkalmazást. A szerver feladata, hogy a hozzá intézett RESTful kérésekhez választ biztosítjon. A REST (REpresentational State Transfer) egy jól meghatározott szabványt biztosít webes hálózatban való kommunikációval kapcsolatos folyamatokra.

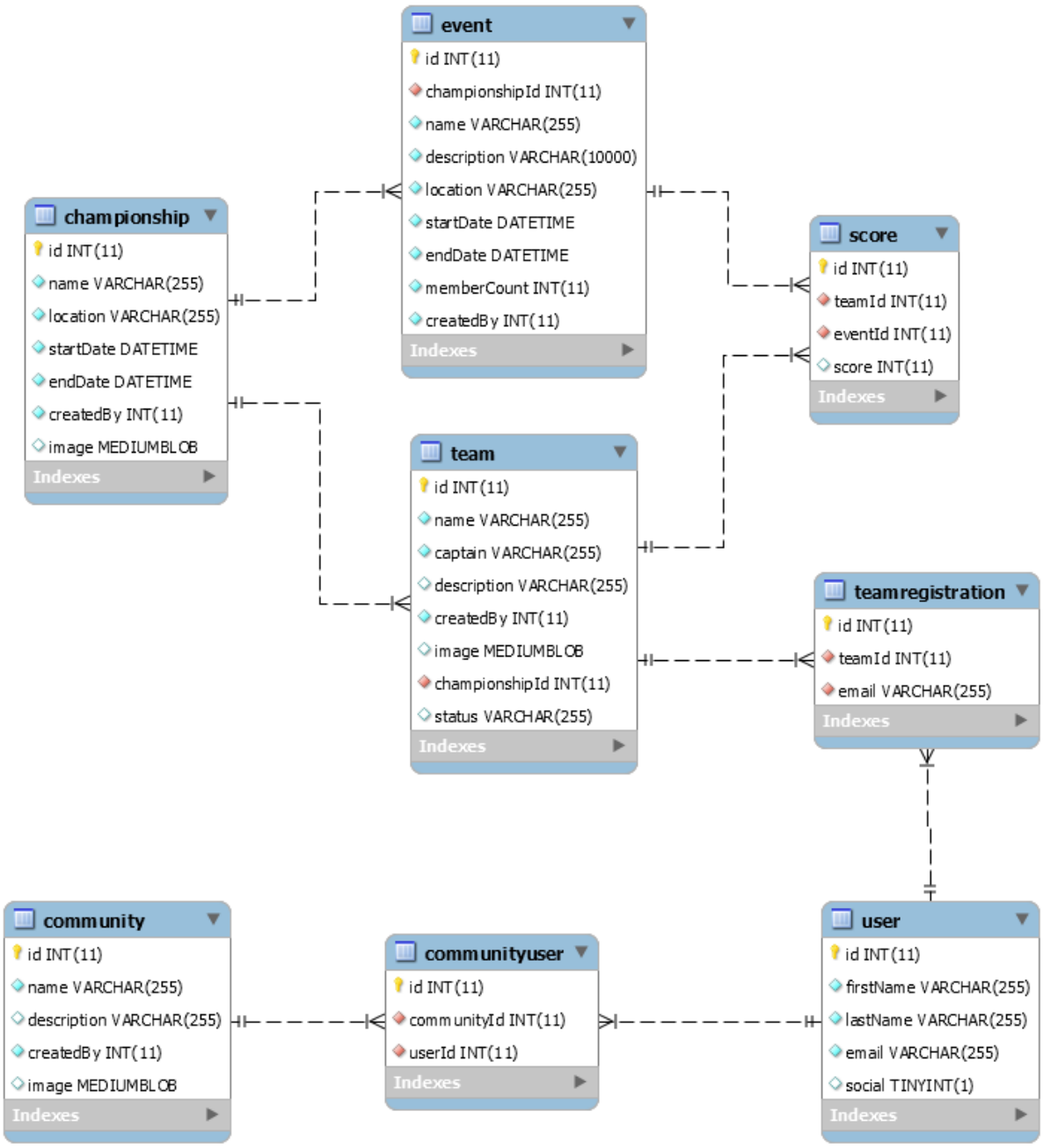


2. ábra. A DiáknApp szofverrendszer elemei és a közöttük lévő kapcsolatok

A szerveralkalmazás JavaScript [20] nyelvben van implementálva és NodeJS [44] JavaScript futási környezetben van futtatva.

2.1.1. Adatmodell

Az adatokat reprezentáló osztályok a `db/models` csomagban találhatóak. Amint a 3. ábra is mutatja az entitások központjában a `Championship` és a `User` adatmodell áll. A Championship entitásban tárolunk közvetlen információkat a bajnokságokról. A Team entitás a bajoksághoz tartozó csapatokat tartalmazza, valamint azoknak részletes információit. Minden bajnoksághoz több csapat és esemény tartozik. Az eseményekre minden csapat pontot kap, ezt a célt szolgálja



3. ábra. A központi szerver entitásai

a `Score` adatmodell.

Egy másik központi entitás a, `User`, melyben a felhasználók részletes információi vannak eltárolva. Fontos kiemelni, hogy a jelszavak nem itt vannak elmentve, ezt az információt a hitlesítési szerver tárolja. A `User` entitás tartalmaz egy `social` adattagot, amely azt határozza meg, hogy a felhasználó standard vagy Facebook bejelentkezéssel lépett be a rendszerbe. Ezt azért szükséges elmenteni, hogy a megfelelő hívást tudjuk küldeni a hitelesítési szervernek.

A felhasználók csapatokba tartozhatnak, amelyet a `TeamRegistration` entitás reprezentál. Ugyanaz a felhasználó akár több csapathoz is tartozhat, mivel különböző bajnokságokon különböző csapatokban indulhat. Továbbá a felhasználók közösségekhez is tartozhatnak, amellyel egységesen indulhatnak csapatként a bajnokságokon. Ebben az esetben nincs megszabva, hogy a közösség minden tagjának részt kell vennie a bajnokságban.

2.1.2. Adatelérési réteg

A központi szerver az adatokat egy MySQL [16] relációs adatbázisban tárolja el. A táblák definíciója a `db/models` csomagban találhatóak. Itt vannak definiálva az entitások közti kapcsolatok, illetve azokra nézve a különböző megszorítások, alapértelmezett értékek és további tulajdonságok, amelyek az adatbázis világában érvényesülnek.

A központi szerver az adatbázissal való kommunikációt a Sequelize [7] könyvtár segítségével valósítja meg. A Sequelize egy *promise*-okon [36] alapuló objektumok relációs leképezését (ORM) [17] biztosító könyvtár.

Az adatbázisban szereplő adatokat lekérdező és módosító függvények a `db/queries` csomagban található fájlokban vannak implementálva. Itt minden entitáshoz társítva van egy JavasScript lekérdező fájl, amely az adott entitással kapcsolatos lekérdezéseket határoz meg. Az adatmodellhez hasonlóan ezek a lekérdezések is teljesen adatbázis független módon vannak megadva. A lekérdező fájlokban `CRUD` műveletek találhatóak, amelyek csak az adott entitásra vonatkoznak.

2.1.3. Biztonság

Az alkalmazás használatba vételéhez, legyen ez web- vagy mobilalkalmazás esetén, a felhasználónak nem szükséges bejelentkeznie, de hogy ki tudja használni a rendszer által nyújtott teljes funkcionalitásokat, bejelentkezés szükséges. Az 1. fejezetben bemutatott szerepkörökhöz különböző jogosultságok tartoznak, és annak érdekében, hogy minden felhasználó a neki szánt tartalmat érje el, ellenőrizni kell, hogy be van-e jelentkezve, illetve, hogy jogosult-e az adott kérés végrehajtásához.

A web- és a mobilalkalmazások esetén egyaránt meg lehet különböztetni a bejelentkezett és a nem bejelentkezett felhasználókat. A bejelentkezéshez az API szerver egy OAuth [15]

provider-ként szolgáló hitelesítési szervert használ, amelyet nem a dolgozat írói fejlesztettek. Az OAuth egy engedélyezési protokoll, ami lehetővé teszi a felhasználóknak azonosítását az alkalmazásokon belül. Az engedélyezést JSON Web Token-ek (JWT) [23] segítségével valósul meg. A bejelentkezés két módon történhet meg: standard, illetve Facebook-os bejelentkezés által.

Standard bejelentkezés esetén a felhaszanáló a regisztráció során megadott adatokat adja meg, feltéve, ha a regisztráció sikeres volt. A kliensek minden standard bejelentkezéshez kötelezően el kell küldeniük a hitelesítési szerver által biztosított kliens azonosítókat, amelyek mindhárom kliens esetén különbözőek, ám egyaránt titkosak.

Első lépésként, a szerver ellenőrzi az adott e-mail cím létezését a 3.1. fejezetben tárgyalt adatbázisban. Amennyiben az e-mail cím nem létezik, a bejelentkezés nem tud megtörténni. Ha az e-mail létezik, a szerver továbbítja az adatokat a hitelesítési szervernek, amely leellenőrzi, hogy az adott e-mail cím és jelszó páros jelen van-e a saját adatbázisában, illetve helyes-e. Amennyiben a folyamat sikeres, hitelesítési szerver egy JWT tokent térít vissza a API szervernek. Ezeknek a személyes információka t(mint az e-mail cím és jelszó) tartalmazó tokeneknek van egy előre meghatározott lejárási idejük. A API szervernek vissza kell küldenie a JWT tokent és a felhasználói azonosítót a klienseknek. Ezzel befejeződik a bejelentkezés, és minden olyan kérés, amely jogosultságokat igényel egy, a kérés fejlécébe hozzátett token segítségével fog megtörténni.

Facebook-os bejelentkezés során a webböngésző előbb átirányít a Facebook oldalra, a telefonkészülék pedig a telepített Facebook alkalmazásra, hogy a felhasználó előbb bejelentkezhessen a fiókjába. Amennyiben a felhasználó korábban már bejelentkezett az adott készüléken, adatai pedig mentésre kerültek, újabb bejelentkezés esetén csupán a folyamat megerősítése szükséges.

Ezután a kliensek jogosultságokkal kapcsolatos kérést küldenek a Facebook-nak, a Facebook pedig visszaküld nekik egy tokent, amely az előbbiekben említettekhez hasonlóan személyes adatokat tartalmaz. Ezt a tokent a kliens elküldi az API szervernek, amely a token segítségével egy új kérést küld a Facebook-nak (a szerver kéri le a felhasználóval kapcsolatos adatokat és menti le a token alapján).

Az API szerver ezek után leellenőrzi, hogy létezik-e már az adatbázisban a Facebook által visszaadott e-mail címhez tartozó felhasználó. Amennyiben ilyen felhasználó még nem létezik, létrehozza, vagyis regisztrálja azt, és hozzáadja a munkamenethez a kapott tokent, a felhasználó azonosítóval együtt. Ha már létezik felhasználó az adott e-mail címhez, akkor a regisztrációra nyilvánvalóan nincs szükség, csupán a tokent és a felhasználó azonosítót kell hozzárendelni a munkamenethez, az API szerver azonban mindkét esetben visszaküldi a kliensnek a kapott tokent és a felhasználó azonosítót.

A felhasználók szabadon használhatják a szerver által biztosított eléresi útvonalakat minden bejelentkezés után. A bejelentkezésből még nem derül ki, hogy az adott felhasználónak van-e jogosultsága elvégezni egy adott műveletet a szerepköre alapján. A felhasználók szerepköreinek megállapítását egy szerepkör alapú hozzáférést szabályozó rendszer végzi (RBAC). Ezen belül minden felhasználó számára meghatározott, hogy milyen feltételek mellett milyen műveleteket tud elvégezni.

Az RBAC létrejöttekor a konstruktorban létrejön egy `roleMap` hasító tábla, amely segítségével megállapítható, hogy az adott paramétérek alapján a bizonyos szerepkörből való felhasználó jogosult-e elvégezni a kért műveleteket. Ennek érdekében az RBAC rendszer biztosít egy `can` nevű függvényt, amely három paramétert kap bemenetként (szerepkör, végrehajtandó művelet, paraméterek), amelyek alapján eldönteni, hogy a felhasználó elvégezheti-e az adott műveletet vagy sem.

2.1.4. API

A DiáknApp projekt keretein belül működő web- és mobilalkalmazásokat egy API szerver szolgálja ki, amely egy RESTful (Representational State Transfer) [30] és nyilvános szolgáltatásként üzemel. Ez szolgálja ki a kliensek által küldött HTTP kéréseket. A szerver és a kliensek által folytatott kommunikáció HTTP protokollokon alapszik, és JSON formátumú üzenetekkel valósul meg. Az API szerver legfontosabb elérési pontjai a következőkben kerül bemutatára.

A bajnokságokért a `/championships` elérési útvonal felel, amely az alkalmazás központi része. Ezen belül számos dologra adódik lehőség: listázhatóak a már meglévő bajnokságok, létrehozhatóak újak. A `/championships/:id` lehetőséget ad, hogy az adott egyedi azonosítóval rendelkező bajnokságot módosítsák, töröljék, vagy lekérjék. Az előbbi kettő csak abban az esetben hajtható végre, ha a felhasználó eleget tesz az RBAC által biztosított `can` nevű függvénynek, illetve ő maga hozta létre az adott bajnokságot.

A `/:championshipId/events` elérési pontnál lekérdezhetőek az adott bajnoksághoz tartozó eseményeket, törölhetőek és modosíthatóak azok, abban az esetben, ha az adott felhasználó volt, aki létrehozta azokat. A `/:championshipId/events/:eventId` elérési pontnál ugyanazok a műveletek végezhetőek el, mint a `/:championshipId/events`-nél, csak itt egy eseményre végezhetőek el a kért műveletek, és nem az összesre.

A `/:championshipId/teams` elérési pontnál egy adott bajnoksághoz tartozó csapatok menedzselése végezhető el. A `/:championshipId/teams/:id/members` elérési pont a csapattagokkal kapcsolatos műveletekért felel, illetve `/:championshipId/teams/:id/` elérési pontnál a CRUD műveletek végezhetőek el az adott csapatra. A `/:championshipId/:teamId/:status` elérési pontnál a bajnokságot létrehozó felhasználó

eldöntheti, hogy az illető csapat részt vehet-e a bajnokságon vagy sem, állítva a csapat státuszát. A `/:championshipId/scores` elérési pontnál lekérhetőek a csapatok pontjai, amelyeket a bajnokságért felelős felhasznaló módosíthat.

A bejelentkezésért és a kijelentkezésért a `/users/login` elérési út a felelős. Facebook-os bejelentkezés esetén `/users/sociallogin` elérési pont fogadja a kéréseket.

A közösségekért a `/communities` elérési út felel, és a `/championships`-hez hasonló műveletek végezhetőek el. A `/communities/:id` elérési pont visszatéríti az adott azonosítójú közösséget, illetve módosítani és törölni is tudja az azért felelős felhasznaló. A `/communities/:id/users` elérési pontnál lekérhető az adott azonosítójú közösséghez tartozó összes felhasználó, illetve újak adhatóak hozzá. A `/communities/:id/users/:userId` elérési pontnál a közösséget létrehozó felhasználó eltávolíthatja az adott azonosítóval rendelkező felhasználót a közösségből.

2.2. A web kliens

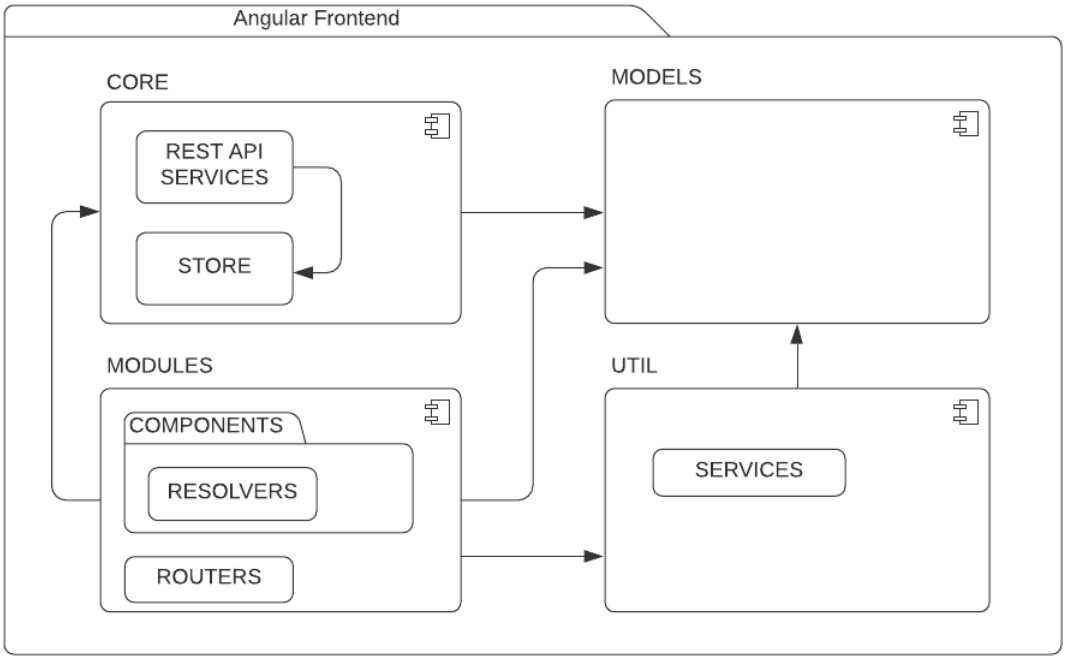
A webalkalmazás Angular [42] keretrendszerre épül, TypeScript-ben [27] íródott és négy fő részből áll: `core`, `modules`, `util` és `models`. Ezek a csomagok egymással kommunikálnak, így biztosítják az alkalmazás működését (lásd a 4. ábra).

A `models` csomag tartalmazza a kliensoldali entitások modelljeit. Ezek a modellek nem teljesen egyeznek meg a szerveroldali entitásokkal, tartalmaznak olyan plusz mezőket is, amelyeket a felhasználó adott meg. Ilyen mező például a csapattagok listája, amely egy csapat létrehozásának az egyik fontos eleme.

A `core` csomag tartalmazza a szerverrel való kommunikációs szolgáltatások implementációit (Rest Api Services), valamint az alkalmazás jelenlegi állapotát tároló implemetácókat. Az alkalmazás állapotán értjük a bejelentkezett felhasználó adatait, mint például a személyre szabott tokent, amelyet a szerver biztosít minden felhasználónak (lásd 2.1.2. fejezet). Ebben a modulban történik meg a szervertől kapott adatok automatikus konverziója a *models* csomagban definiált modellekre.

A `modules` csomagban vannak definiálva a felhasználói felülethez szükséges komponensek, és ezeknek az elérését biztosító routerek. Minden komponenshez tartozik egy resolver, amely biztosítja az adatok helyességét, hogy ne jelenjenek meg olyan problémák, mint a rossz adatok megjelenítése.

A `util` csomagban olyan segéd szolgáltatások találhatóak, amelyeket a komponensek sok helyen is használnak, mint például a képfeldolgozás, vagy a dialógus ablakok megjelenítése.



4. ábra. A webalkalmazás architektúrája és a csomagok közti függőségek

2.2.1. Kommunikáció a szerverrel

A szerverrel való kommunikáció HTTP protokollokon keresztül történik, a Fetch API [18] használatával. A kliens aszinkron kéréseket küld a szerver felé, majd visszahívó függvények kezelik le a választ. Az observable tervezési minta alapján a szolgáltatások eljuttatják a választ a megfelelő komponenseknek.

Minden, a szerver felé küldött és a szervertől kapott kérést interceptorok kapják el és módosítják azokat. Az interceptorok implementálják az Angular `HttpInterceptor` interfészt. A szerverhez küldött kérésekhez hozzácsatolja a kliens azonosítóját, amelyre a szervernek van szüksége, hogy azonosítsa a megfelelő klienseket. Ezen kívül, ha egy felhasználó be van jelentkezve, akkor a kérésekhez hozzácsatolja a felhasználó egyedi tokenjét is. A szervertől kapott válaszokat is egyszer egy interceptor kezeli le, csak ezután jutnak el a szolgáltatásokhoz. Ahogyan az 1. kódrészletben is látható, egy *middleware* lekezeli azt az esetet, amikor a szervertől 401-es hibakód érkezik, vagyis érvénytelen a felhasználó tokenje, viszont a kliensoldali token még érvényben van. Ilyenkor a kliensoldali token érvénytelenítődik, és úgymond kijelentkezik a felhasználó a weboldalról.

A bejelentkezett fehasználó adatai `localStorage` konténerben vannak tárolva, amelyek az alkalmazás állapotát hivatott tárolni a böngészőben, kulcs-érték párok formájában. Jelenleg az alkalmazás állapota a bejelentkezett felhasználó *id* és *token* azonosítóiból áll. A felhasználók adatait kezelő szolgáltatás tudja módosítani, illetve törölni ezeket a mezőket, a felhasználó állapotától függően.

```
intercept(request: HttpRequest<any>, next: HttpHandler):
    Observable<HttpEvent<any>> {

    return next.handle(request).pipe(
      catchError((err) => {
        if (err.status === 401 && this.tokenStore.hasToken()) {
          this.tokenStore.deleteToken();
          this.userStore.deleteUserId();

          this.userService.authenticationListener.next(
              new AuthenticationData(false, null)
          );

          location.reload();
        }

        return throwError(err);
      })
    );
 }
```

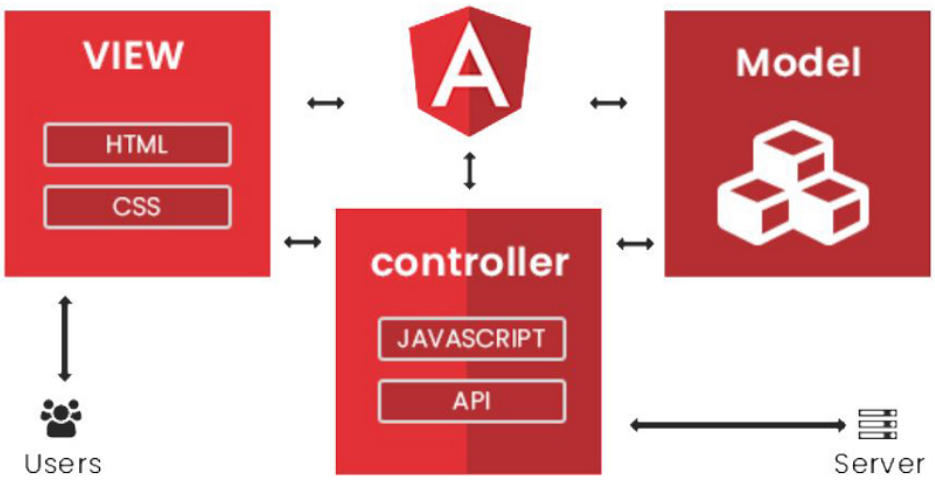
1. kódrészlet. A szerver válaszát feldolgozó interceptor

2.2.2. Komponensek közötti (belső) navigáció

A belső navigáció az `Angular Router` modul felhasználásával történik. A routerek segítségével elkülöníthető, hogy melyek azok a komponensek, amelyek láthatóak kell legyenek egy adott elérési útvonalon. Emellett a routerek közötti navigáció lehetővé teszi a nézetek közti könnyű váltást, amelyet egy felhasználó többféleképpen is elérhet:

- egy adott URL cím megadása a címsorban
- a weboldalon megadott linkek segítségével
- a böngésző előre és hátra gombjainak a segítségével

Minden komponenshez tartozik egy elérhetőségi útvonal a routereken keresztül, valamint egy resolver, amely az adatok betöltéséért felel. Mivel a szerverhez küldött kérések aszinkron módon történnek, a böngésző nem várja meg a választ, hanem egyből megjeleníti a komponenseket, esetenként hiányos adatokkal. Ebből az okból kifolyólag szükség volt `resolver`-ek használatára. Egy adott oldal eléréséhez megadhatóak különböző `resolver`-ek, amelyek nem engedik a böngészőnek, hogy megjelenítse az adott komponenst, amíg nem érkezett válasz a szervertől. Ily módon biztosítja a megjelenített adatok helyességét.



5. ábra. MVC elv megvalósítása az Angular környezetben

2.2.3. Megjelenítési réteg

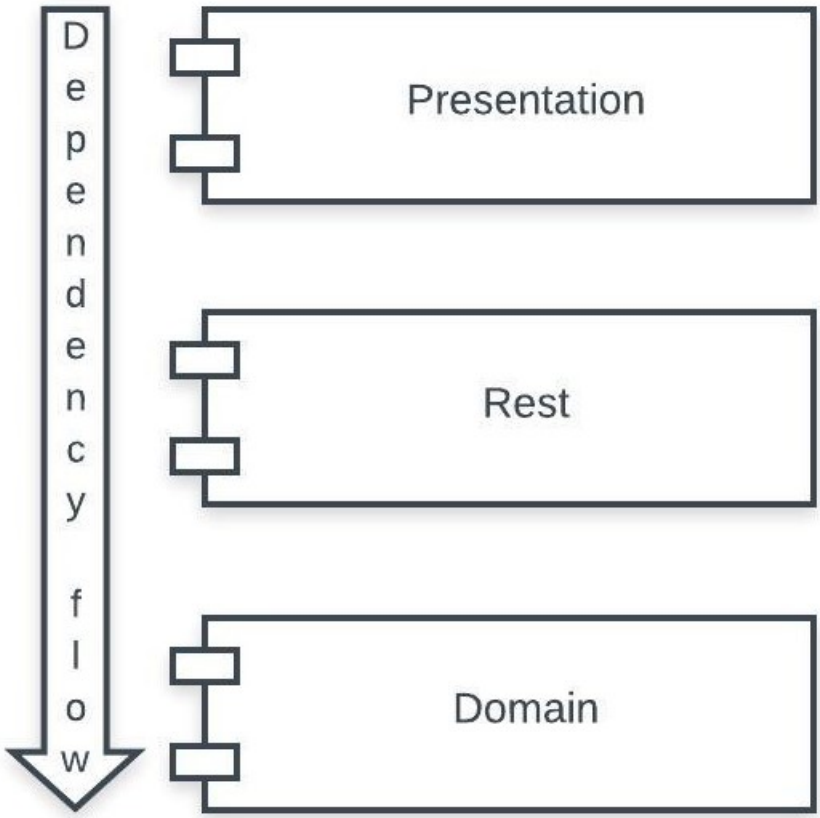
Az Angular környezet biztosítja az MVC (Model-View-Controller) elv megvalósítását, ahogyan az 5. ábra[1] is mutatja. Az alkamazás `models` csomagja tartalmazza azokat az adatmodelleket, amelyeket a felhasználó böngészhet. A megjelenítés nem direkt módon történik, hanem a kontroller feldolgozza ezeket az adatokat, majd átadja a nézeteknek. A TypeScript állományok töltik be a kontrollerek szerepét az applikációban, ezek végzik el az üzleti logikát. A HTML és CSS állományok biztosítják a felhasználói felületet, amellyel a felhasználó interakcióba léphet, egy gombnyomás vagy egy kattintás segítségével. Ezáltal események váltódnak ki, amelyet a kontrollerek kezelnek le, esetenként a modellekkel vagy a szerverrel kapcsolatba lépve. Hogyha szükséges akkor a műveletek elvégzése után frissítik a felhasználói felületet.

A felhasználói felület az Angular Material [13] könyvtár elemeinek segítségével van felépítve. A könyvár biztosít előre definiált témákat is, viszont megadja a lehetőséget, hogy egyedi témákat is lehessen definiálni az alkalmazásokban. A DiáknApp témája Sass [47] nyelvben van definiálva, Material színek és paletták segítségével. Egy témának a létrehozásához definiálni kell egy elsődleges és másodlagos színt, valamint opcionálisan megadni más alap színeket is, mint a felhívó szövegek színe, vagy a hibák színe.

2.3. Az Android kliens

A DiáknApp szoftverrendszer a megszervezett események nyomonkövetése érdekében biztosít egy Android alkalmazást, amely a JetBrain által fejlesztett Kotlin [24] nyelvben írodott.

[1] Forrás: `https://wuschools.com/what-is-mvc-and-understanding-the-mvc-pattern-in-angular/`, elérés dátuma: 2020. 04. 25



6. ábra. Android alkalmazás architektúra diagramja

Az alkalmazás a Clean architektúra [29] elveit valósítja meg. A Clean architektúra a projekt szerkezetére van hatással, ugyanis a projektet több rétegre osztja. Meghatározza a kód osztályokba és fájlokba való rendezését, az osztályok közötti kapcsolatokat, valamint különálló komponensekre osztja a kódot, ezáltal könnyen karbantarthatóvá válik a projekt, hiszen ha megváltozik valamelyik komponens implementációja, az nem vonja maga után más komponens implementációjának változtatását. Egyik alapvető elve, hogy a belső rétegek nem tudnak az alkalmazás külső rétegeiről, tehát a függőségek befelé irányulnak.

A rétegek közzéppontjában az adatelérési réteg és a központi entitások foglalnak helyet. Ezek köré épülnek a használati esetek (*use case*-ek), amelyek az entitásokhoz vannak társítva, pl: `ChampionshipModel` - `ChampionshipUseCase`. A harmadik réteg a megjelenítési réteg, amely a felhasználói felület megjelenítéséért felelős. A külső réteg, maga a képernyő, amely a felhasználói interakciókat érzékeli. Ezáltal, egy gombnyomás során a rétegek sorra hívják az egyel bennebb található réteget: a felhasználói interfész érzékeli a gombnyomást, a prezentációs réteg értelmezi azt, és egy hívást társít hozzá, amelyet a használati eset továbbít az adatelérési rétegnek. Az adatelérési réteg végül az alkalmazásszervernek intéz egy HTTP kérést, majd a választ szintről-szintre lebontja.

Az applikáció három fő modulból tevődik össze: `presentation`, `domain` és `rest` (lásd

6. ábra). A hierarchia legalján található a domain modul, amely a másik két modultól teljesen független, valamint nincs más függősége sem. Itt találhatóak a központi entitásokhoz tartozó POJO-k és a rendszer funkcionalitásaihoz tartozó interfészek. A domain réteg fölött a `rest` modul található. Ezen a szinten található a domain rétegben definiált interfészek implementációja, valamint ez a réteg gondoskodik a szerverrel való kommunikációról is. A legfelső réteg, a `presentation`, amely a felhasználói felület megjelenítéséért felelős. Ehhez a réteghez tartoznak a kinézetek, a hozzájuk társított kontrollerek és View Model-ek is.

A három modul mellett található még egy negyedik, `util` modul, amely különböző, a többi modultól független funkciókért felel. Itt megtalálható egy `Image` osztály, amelynek feladata a képeket *base64* formátumból `Bitmap`-é alakítása és vissza. Továbbá megtalálhatóak még a `UserStorage`, `TokenStorage` és `Storage` osztályok, amelyek a felhasználó adatait és a hitelesítési tokent kezeli. Ezek mellett megtalálható még egy `DateFormatter` nevű statikus osztály, ami a nevéből adandóan dátum formázásokat végez. Erre azért volt szükség, mivel a képernyőn több helyen, több fragmensen is dátumokat jelenít meg az alkamazás, hasonló formátúmokban, így nem fordul elő kódismétlés, mivel csak ennek az osztálynak a formázó eljárásai kerülnek felhasználásra.

2.3.1. Adatelérési réteg, kommunikáció a szerverrel

Az alkalmazás REST hívásokkal kommunikál a központi szerverrel. A HTTP kérések elküldése és fogadása Retrofit [6] segítségével vannak megvalósítva. A Retrofit egy programkönyvtár, amely típusbiztos HTTP klienst biztosít Android alkalmazásoknak. Nagyban megkönnyíti a hálozati kérések implementációját, mivel könnyen hozzáadhatunk egyedi

```
companion object {
    fun create(): RetrofitEventRestClient {
        val client = OkHttpClient.Builder()
            .cookieJar(SessionCookieJar)
        client.interceptors().add(AuthInterceptor())

        val retrofit = Retrofit.Builder()
            .client(client.build())
            .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
            .addConverterFactory(GsonConverterFactory.create())
            .baseUrl(ApiBase.getBaseUrl())
            .build()
        return retrofit.create(RetrofitEventRestClient::class.java)
    }
}
```

2. kódrészlet. HTTP kérést létrehozó statikus osztály

fejléceket, kérés típusokat, fájlokat tölthetünk fel, ami által lerövidíti a kódot. A Retrofit számára kell definiálni egy POJO model osztályt, amely jelen esetben a DTO osztályok, valamint egy interfészt, ami által létrehozza a HTTP kérést. Az interfészen belül annotációk segítségével személyre szabhatókak a kérések: metódus megadása (`POST`, `PUT`, `DELETE`), fejléc hozzáadása, URL paraméterek hozzáadása. Hogy a kérés megvalósuljon, létre kell hozni egy `Retrofit.Builder` instanciát. Ez hozza létre a tényleges HTTP kérést, amelynek meg kell adni az elérési útvonalat, ahová a kérést küldi (lásd 2. ábra). A válaszként érkező adatok feldolgozására a Google által fejlesztett Gson Converter-t [37] alkalmaz. A Gson Converter a JSON formátumban érkező adatokat a fentebb említett POJO objektummá alakítja.

A válaszok fogadására és feldolgozására RxJava [35] könyvtárat alkalmaz. Az RxJava aszinkron kérések feldolgozását segíti, illetve a végrehajtási szálak kezelését teszi lehetővé. Az RxJava három visszatérítési típust biztosít, amelyek az `io.reactivex` csomagban találhatóak meg: `Single`, `Completable` és `Maybe`. A Single olyan `Observable`-ből származtatott típus, amely vagy a modellé konvertált választ, vagy pedig a hibát tartalmazza. Az Observable a `ReactiveX` csomag része és egy megfigyelővel fel lehet iratkozni rá, ezáltal értesít a megfigyelt esemény bekövetkezéséről. A Maybe a Single-hez hasonló, viszont sikeresen végre tud hajtódni úgy, hogy nem tartalmaz adatot a válaszban. A Completable nem tartalmaz adatot, csupán azt jelzi, hogy a kérés sikeres volt-e. A különböző HTTP kérések ezen három típusból térítik valamelyiket, amelyre a prezentációs rétegben (lásd 2.3.2. fejezet) a `subscribe` metódussal fel lehet iratkozni. A `subscribe` két metódust vár paraméterként, az első akkor hajtódik végre, mikor a kérést sikeresen végrehajtódott és a válasz megérkezett, a másodikat pedig mikor hibával tért vissza a válasz. A hibaüzenetben megtalálható a hiba pontos leírása.

A kérésekhez továbbá hozzá van fűzve egy interceptor, amely minden kérés esetén a fejlécben beállítja az `Authorization` mezőt. A cookie-kat is szükséges interceptorhoz hasonlóan beállítani, mivel minden elérési útvonalhoz külön HTTP kliens tartozik, így nem őrzi meg a hitlesítési tokent és a felhasználó azonosítóját, amelyek a szerver hitelesítési folyamataiban szükségesek. Ennek érdekében lett bevezetve egy `CookieJar` osztály, ami az interceptorhoz hasonlóan elmenti a válaszból a cookie-kat, valamint a kérésekhez hozzáfűzi azokat.

2.3.2. Prezentációs réteg

Az alkalmazás esetén az oldalakat azon osztályok képviselik, amelyek az `Activity` osztályból származnak. Viszont nem ezek határozzák meg az oldal elrendezését, csupán az oldalakhoz kapcsolodó logikát tartalmazzák. A nézetet leíró fájl a 2.3.3 fejezetben van részletezve.

Egy adott Activity-n akár több nézetet is megjeleníthetünk, ilyen esetben az Activity csak

helytartó szerepet játszik és fragmensek vannak megjelenítve rajta. Ez azért előnyös, mivel fragmensek esetén gyorsabb és nyomonkövethetőbb a navigáció, ugyanis tartozik hozzájuk egy verem, amely az oldalak közötti váltásokat menti el. Amennyiben egy új fragmens kerül megjelenítésre az előzőleg megnyitottat elmenti a verembe, majd mikor a felhasználó vissza szeretne navigálni az előző oldalra betöltődik a verem tetején található fragmens. Továbbá ezt könnyebenn lehet befolyásolni, mint az Activity-ket, mivel a programozó dönthet arról, hogy hozzá szeretné-e adni a veremhez a fragmenst mikor megnyílik egy új.

A prezentációs réteg a webalkalmazáshoz hasonlóan MVC elvet követ (lásd 2.2.3. fejezet), minden oldalhoz tartozik egy View Model és egy fragmens. A View Model nem tartalmaz logikát, csupán a képernyőn megjelenített adatok tárolására és annak elérésére szolgál. A View Model-ek kérik le a *rest* rétegtől az adatokat, feliratkoznak azok válaszára, majd amint az megérkezik az, értesítik a fragmenseket, amelyek frissítik az oldal tartalmát. Ez megtévesztő lehet, mivel nagyban hasonlít az MVVM [39] tervezési mintára, viszont abban különbözik, hogy MVVM esetén a ViewModel nem tartalmazhat instanciát olyan osztályról, amelynek hozzáférése van az alkalmazás kontextusához és az attribútumai hozzá vannak kötve a képernyőn megjelenített elemekhez. A kontextus egy interfész, amely globális információt biztosít az alkalmazásról. Ennek segítségével hozzáférhetünk osztályokhoz és erőforrásokhoz, így a képernyőn megjelenített elemekhez is.

A fragmensek feladata az események kezelése, mint például gombnyomások és navigáció az oldalak között, továbbá az adatok személyre szabása (pl. dátumok formázássa, megjelenítése és a bemeneti adatok lekérése).

2.3.3. Felhasználói felület

Android alkalmazás esetén a felhasználói felületet kétféle képpen lehet létrehozni: programatikusan és deklaratívan. A DiáknApp alkalmazás esetében keveredik a kettő, viszont többségében deklaratívan vannak megvalósítva a nézetek, és csak esetenként történik rajtuk programatív változtatás.

A deklaratív nézetek leírása egy XML állomány segítségével valósul meg. Ezek az állományok a `res/layout` könyvtárban vannak eltárolva. Minden oldalhoz tartozik egy XML állomány, amely címkék segítségével hierarchikusan felépíti az oldalt. Minden oldal egyetlen gyökér címkét tartalmaz, ez határozza meg, hogy az oldalon hogyan helyezkednek el az elemek: `LinearLayout`, `RelativeLayout`, `FrameLayout` és ConstraintLayout. Az alkalmazásban leggyakrabban használt elrendezések a `FrameLayout` és `ConstraintLayout`.

A FrameLayout csak helytartó, ami egy elemet tud megjeleníteni, ezért alkalmas a fragmensek megjelenítésére. A ConstraintLayout esetében minden elemnek kötelezően kell tartalmaznia két attribútumot. Az egyik azt határozza meg, hogy horizontális irányban melyik

elemhez kapcsolódik és milyen távol van tőle, a másik pedig vertikálisan. Ezáltal az elemeknek nem abszolút pozíciójuk lesz, hanem egymáshoz viszonyított helyzetük, így támogatva a telefonok különböző méreteit és felbontását.

Az alkalmazás felületének vannak olyan részei is, amelyek programatikusan vannak felépítve, vagy épp befolyásolva. Egy ilyen oldal a bajnokságokhoz tartozó eseméyek listázása. Mivel nem minden bajokság ugyanolyan hosszú és lehetnek olyan napok is, amelyeken nincsenek események, ezért nem lehet XML állományban egy általános kinézetet meghatározni, az oldal programatikusan épül fel. Ennek az oldalnak is található egy XML leíró állománya, viszont ez csupán az oldal elrendezését határozza meg. Az oldal felépítésében szerepet játszik egy másik XML állomány is, amely egy adott esemény adatainak elrendezését és kinézetét határozza meg. Ezt az állományt minden esemény esetében rendereli a memóriában a megjeleníteni kívánt adatokkal, majd hozzáadja az oldalhoz.

Az oldalak közötti navigáció megkönnyítése érdekében az alkalmazás tartalmaz navigációs menüket. A fő navigációs menü egy `Navigation Drawer`, amely az alkalmazás főbb oldalait köti össze. Ezen kívül, megtalálható egy alsó navigációs menü is, amely a bajnokságokhoz tartozó oldalakat köti össze. A két navigációs menü külön nézeteken szerepel, ezzel is megkönnyítve az alkalmazás kezelhetőségét.

2.3.4. Domain réteg

A domain réteg magába foglalja az entitások osztályait, valamint a hozzájuk tartozó interfészeket, amelyek a használati eseteket definiálják. Az entitások meghatározására `DataClass`-t alkalmaz, amelynek feladata, hogy tárolja az adatokat. A `DataClass` abban különbözik a közönséges osztályoktól, hogy a fordító autómatikusan társít hozzájuk *equals()*, *hashCode()*, *toString()* függvényeket. Kotlin programozási nyelv esetén nem kell *getter* és *setter* függvényeket írni, így ezek az osztályok POJO-knak felelnek meg.

A modell osztályok mellett megtalálhatóak a felhasználási esetek. Minden entitáshoz tartozik egy külön osztály, amely a felhasználási eseteket tartalmazza (`Championship` - `ChampionshipUseCase`). A felhasználási esetben a hozzá tartozó objektumra vonatkozó műveletek érhetőek el.

2.3.5. Adattárolás

Ahogy korábban említve volt (a 2.1.2. fejezetben), a hitelesítési szerver egy tokent generál. Hogy az Android kliens későbbi felhasználás esetén is megőrízze a tokent, a alkalmazás a *Shared Preferences*-ben tárolja el azt. A *Shared Preferences* perzisztens adatok tárolására alkalmas, az adatokat kulcs-érték párosokként tárolja. Az itt elmentett információk az alkalmazás bezárása után is megmaradnak, így következő induláskor a program

könnyedén kiolvashatja azokat. Az adatok kezeléséről a `TokenStorage` és `UserStorage` osztályok gondoskodnak. Ezen osztályok hívják a `Storage` statikus osztály függvényeit, amely közvetlenül a `SharedPreferences`-el áll kapcsolatban. Az adatok egy általunk elnevezett titkosított fájlba vannak elmentve, és a biztonság érdekében a hozzáférést *MODE_PRIVATE*-ra van állítva. Ez azt eredményezi, hogy az elmentett adatokat csak a DiáknApp alkalmazás tudja módosítani.

2.4. Az iOS kliens

A DiáknApp szoftverrendszer tartalmaz egy iOS alkalmazást is, amely a megrendezésre kerülő események nyomkövetését és menedzselését biztosítja. Az iOS mobilalkalmazás feladata a szerverrel való kommunikáció, illetve az iOS felhasználói felület üzemeltetése. A DiáknApp iOS alkalmazása az Apple által fejlesztett Swift [28] programozási nyelvben íródott.

Az iOS alkalmazás MVC architekúrán alapszik. A modell és a nézet közvetlenül nem kommunikálnak egymással, csak a kontrolleren keresztül. A modell képviseli a sajátos adatokat az alkalmazáson belül, mint például a bajnokság, csapat stb. A nézet a megjelenítésben játszik szerepet, általa tudja kezelni a felhasználó az alkalmazást.

2.4.1. Kommunikáció a szerverrel

Az iOS mobilalkalmazás, az Android és webalkalmazáshoz hasonlóan, REST kéréseken keresztül kommunikál a központi szerverrel, amely az adatelérési rétegen van implementálva. Az alapvető HTTP kérések fejléceinek és törzseinek összeállításáért az Alamofire [11] felel. A könyvtár megkönnyíti a kérések és válaszok implementálását iOS környezetben. A kérések implementációi a `Network` könyvtárban, különböző fájlokban találhatóak. A RestManager osztályban a metódusok feloszthatóak annak függvényében, hogy milyen típusú kérést szeretne a felhasználó végrehajtani, illetve a kérések formátumának típusa milyen legyen, amely az alkalmazás esetében `application/json`. Egy ilyen esetet mutat be a 3 .kódrészlet.

A Network könyvtár többi állománya (`AuthRestManager`, `EventRestManager`, `TeamRestManager`, stb.) kibővítik a `RestManager`-t, elkülönítve ezáltal a modellekhez tartozó

```
func GET(path: String, completion: @escaping (_ data:Data?, _ error: Error?)
-> Void) {
        var request = URLRequest(url: createURL(path: path))
        request.httpMethod = "GET"
        execute(request: request, completion: completion)
    }
```

3. kódrészlet. Általános `GET` kérés.

kéréseket a szerverrel való kommunikáció során. Minden entitás a kérések folyamán átadja a szükséges paramétereket illetve az elérési címet a `RestManager`-nek, amely az általános kéréseket biztosiítja egy adott modellnek. Ez nagyban megkönnyíti a felmerülő hibák javítását, valamint jól elkülöníti a modellek szerverrel való kommunikációját.

Ahogy a 3. kódrészlet is mutatja, minden szerverrel való kommunikációt az `execute` metódus hajt végre, ez a hívások legfelső foka, mivel több hívás lesz egybeágyazva. Itt csatolja hozzá a mobilalkalmazás a kérésekhez a felhasználó tokenjét, ami minden híváskor szükséges, ha be van jelentkezve. Ezáltal a szervernek lehetőségében áll eldönteni, hogy a felhasználó jogosult-e az adott műveletre, vagy sem. A válaszok már `execute` függvényhívásban feldolgozásra kerülnek, szűrni tudja őket az API szerver válaszai szerint, majd visszahívó függvények hívásaával visszatéríti az adatot, vagy hiba esetén a hibát. Sikeres hívás esetén a `Decoder` segítségével átkonvertálja a kapott adatokat a nekik megfelelő modellé. Ennek érdekében DTO tervezési mintát használ a rendszer. Ezután egy újabbb visszahívó függvény visszatéríti az imént átalakított modellt a kontrollernek.

Bizonyos műveletek nem bejelentkezett felhasználó számára is elérhetőek, de a rendszer által nyújtott teljes funkciónalitás csak akkor használható ki, ha regisztrál és bejelentkezik. Minden bejelentkezés előtt a kliens, a kéréséhez kötelezően hozzá kell csatoljon egy szigórúan titkos kliens azonosítót, melyet egy hitelesítési szervertől kap meg. Az adott azonosító ismerete nélkül nem lehetséges a bejelentkezés. A token meg az azonosító `Authorization` fejlécben vannak csatolva.

2.4.2. Adattárolás

A szerverrel való kommunikáció során a 2.1.2. fejezetben került tárgyalásra, hogy a bejelentkezett felhasználók egy tokent kapnak a szervertől, amit az iOS mobilalkalmazásnak el kell menteni, hogy mindig gyorsan és könnyedén el tudja dönteni, hogy egy adott felhasználó be van-e jelentkezve. Az mobilalkalmazás a `UserDefaults` tárolóra támaszkodik, így a teljes futás alatt bárhonnan ismerni az alkalmazás a felhasználóhoz tartozó tokent. Kijelentkezés esetén szintén az említett tároló segítségével tudja törölni azt. A UserDefaults egy `.plist` kiterjesztésű állomány, az alkalmazás fájlrendszerében, és egy hasítótábla adatszerkezethez hasonlóan kulcs-érték páros formályában tárolja az adatokat.

2.4.3. Megjelenítési réteg

Az iOS mobilalkalmazás esetében `View Controller`-ek bíztosítanak alapot az alkalmazás belső szerkezetének. Minden `View Controller` a 2.4.4. fejezetben bemutatott egy nézet megjelenítéséért felel. Amellet, hogy kezelei a felhasználói felületet és az adatok megjelenítését, a `View Controller` kommunikál az alkalmazás más kontrollerjeivel is.

A DiáknApp projekt több ilyen `View Controller`-ből épül fel (`LoginViewController`, `TeamsViewController`, `EventsViewController`, stb.), a különböző kontrollerek különböző nézetekhez tartoznak és egy `View Controller` életciklusa több szakaszból áll (lásd a 7. ábra[2]):

- `viewDidLoad()` - ez a metódus hívódik meg akkor, amikor a nézet teljesen betöltődött. Itt többnyire a szerverrel való kommunikáció valósul meg, vagy olyan függvények meghívása, amiket a nézet betöltése után csak egyszer hajtódnak végre.
- `viewWillAppear()` - ez a metódus meghívódik minden alkalommal, a nézet előtérbe kerülése előtt. Itt még nincs megjelenve a nézet, ezért megjelenés előtt változtathatunk rajta (például komponensek elrejtése).
- `viewDidAppear()` - ez a metódus meghívodik, amikor a nézet megjelent a képernyőn.
- `viewWillDisappear()` - ez a metódus hívodik meg, mielőtt az adott nézet háttérbe kerül.
- `viewDidDisappear()` - ez a metódus akkor hívodik meg, amikor a nézet teljesen háttérbe került.

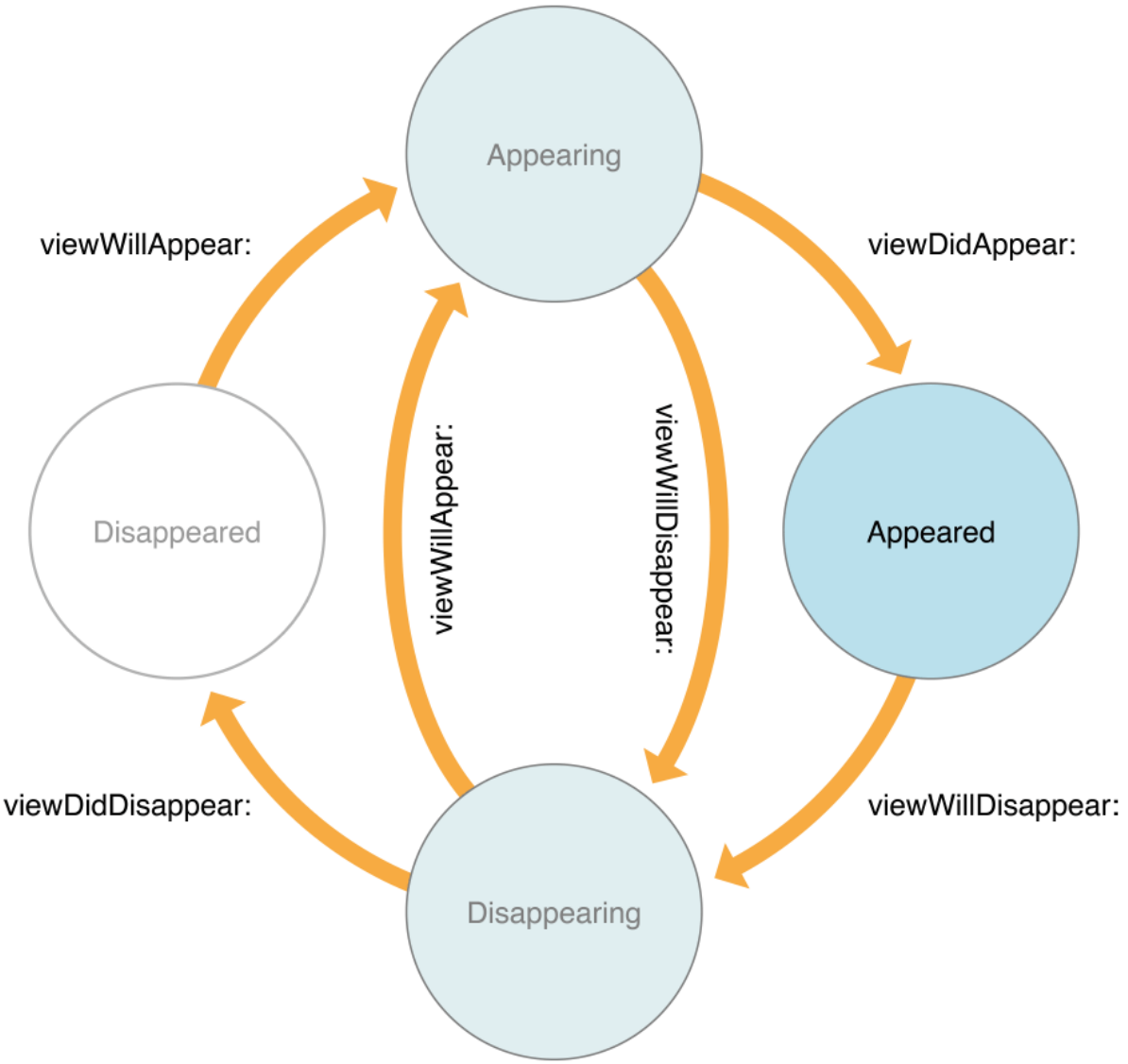
2.4.4. Felhasználói felület

Egy iOS alkalmazás esetében a felhasználói felület kialakítására három módszer áll rendelkezésre, **Storyboard**, **XIB** és programatikus megvalósítas. A DiáknApp iOS alkalmazása esetében megtalálható mindhárom használatra került, de a felületek túlnyomó részét a `Storyboard` bíztosítja.

A `Storyboard` egy XAML [26] leíró állomány, amely a nézetek leírását, az alkalmazás oldalait és a köztük lévő átmenetet tartalmazza `segue`-k segítségével. Navigáció során a nézetek egy "navigációs verem"-ben vannak eltárólva és a legfelső elem jelképezi azt, amit éppen lát a felhasználó. A többi nézet ezáltal el van fedve, és csak akkor kerül előtérbe, mikor a felette lévő nézetek háttérbe kerülnek. Helyenként programatikusan van menedzselve a navigációs verem, a felhasználó interakció függvényében, a szükséges paramétereket továbbítva a megjeleníteni kívánt nézetnek.

Minden nézethez tartozik egy kontroller is, ami a 2.4.3. fejezet részletesebben bemutat. Minden *Storyboard*-hoz tartozik egy vizuális szerkesztő, amely lehetővé teszi, a gombok, szövegdobozok és különböző komponensek hozzáadását a felhasználói felülethez. Ezeknek az komponenseknek az elhelyezkedését, kapcsolatát más elemekkel, stílusát és egyéb tulajdonságait is erről a felületről lehet menedzselni. Ezen kívül a *Storyboard* lehetővé teszi, hogy az objektumok hozzácsatolását az oldal kontrollerjéhez, ahol programatikusan

[2]Forrás: https://medium.com/good-morning-swift/ios-view-controller-life-cycle-2a0f02e74ff5, elérési dátuma: 2020. máj. 2



7. ábra. A kontroller életciklus metódusai, és az azok közötti kapcsolatok.

is befolyásolható azok működése, olyan esetekben, mint például amikor el akar rejteni egy gombot, vagy egy komponens megjelenését akarja változtatni a felhasználó interakciói függvényében.

Az elemek elhelyezkedését az oldalon az `Auto Layout` [40] biztosítja `Constraint`-ek segítségével. Ez annyit jelent, hogy az alkalmazás dinamikusan változtatja az elemek elrendezését a nézeten a megadott `Constraint`-ek függvényében, amelyek meghatározzák a kapcsolatot két elem között a felhasználói felületen. Az `Auto Layout` motor ezen `Constraint`-ek segítségével következtetni ki a méretet és az elhelyezkedésüket a nézeteknek a felhasználói felületen. Az eredmény egy dinamikus felhasználói felület, amely minden eszközön ugyanúgy fog megjelenni figyelembe véve a képernyő méreteit és az állapotát (dőlt vagy portré).

A `Constraint`-ek esetében minden elemnek tartalmaznia kell két attribútumot. Az egyik

horizontális, a másik vertikális irányban határozza meg elemek egymáshoz viszonyított helyzetét.

Az alkalmazás felületének vannak olyan részei is, amelyek `XIB`-ek (XML Interface Builder) segítségével valósít meg. Ezek az állományok egy egyedi megjelenítést bíztosítanak egy adott komponensnek, és megkönnyíti a komponensek újbóli felhasználást. Az iOS alkalmazás megvalósítása során a különböző listák (például bajnokságok, események) megjelenítésénél a cellák tulajdonságai és felépítése `XIB` állomanyok definiálásával történik. A *Storyboard*-hoz hasonlóan itt is elérhető egy vizuális eszköz, amely lehetővé teszi az komponensek hozzáadását a nézethez.

3. Felhasznált technológiák és eszközök

Az alkalmazás szerver JavaScript-ben van megírva a Node.js futási környezetet felhasználva, amely MySQL relációs adatbázisban tárolja el az adatokat. A szoftverrendszer biztosít három klienst: webes alkalmazást, valamint natív Android és iOS mobilalkalmazásokat. A webes platform Angular keretrendszerre épül, az Android alkalmazás Kotlin, az iOS alkalmazás pedig Swift programozási nyelvekben íródott.

3.1. Szerveroldali technológiák

Node.js

A *Node.js* [44] egy pehelykönnyű futtatási környezet, amely segítségével *JavaScript* programokat lehet futtatni böngésző nélkül. A Node.js a Chrome V8 JavaScript [8] motorra épül. Rengeteg modult tartalmaz, amely a programozók feladatainak megkönnyítését segíti elő. A beépített modulokon kívül megtalálhatóak továbbá külső modulok is, amelyet függőségekként lehet telepíteni. Ilyen esetekben szükség van az alapértelmezett npm[38] csomagkezelőjére. A függőségek listája és további információk róluk a `package.json` állományban vannak eltárolva.

Sequelize

A Sequelize egy ORM keretrendszer, amely a Node.js környezethez lett kifejlesztetve. A keretrendszer több adatbázis dialektust támogat, a projekt fejelesztése során használt MySQL dialektust is. Az adatbázis műveletek JavaScript nyelvben vannak definiálva és a könyvtár lefordítja azokat SQL specifikus parancsokká.

Hitelesítési szerver

Ahogy a 2.1.2. fejezetben is említésre került, a felhasználók azonosítására egy hitelesítési szerver van alkalmazva, ami az API szervernek JWT tokeneket szolgáltat. A kliensek és a hitelesítési szerver között nincs direkt kommunikáció, ez az API szerver által valósul meg. A hitelesítési szerver mindhárom kliensnek egy titkos `base64` karakterláncot biztosít, ami kötelezően minden bejelentkezéskor jelen kell legyen, így tudja azonosítani a klienst. Ha a kliens nem rendelkezik a titkos kulccsal, akkor a hitelesítési szerver nem szolgáltat információt annak, ezálal a felhasználó nem tud bejelentkezni a rendszerbe.

MySQL

A *MySQL* [16] egy nyílt forráskódú, SQL alapú adatbáziskezelő rendszer, amelyet az Oracle fejlesztett. A rendszer teljesen platformfüggetlen és az egyik legnépszerűbb relációs adatbáziskezelő rendszer.

Node package manager

A Node package manager [38], rövidítve `npm` egy függőség kezelő rendszer, amelyet főként JavaScript programozási nyelvhez fejlesztettek ki. Két részből tevődik össze, a számítógépre telepített parancssori kliensből és az online adatbázisból, ahol a csomagok (függőségek) megtalálhatóak. A Gradle-hez hasonlóan itt is található egy leíró állomány, a `package.json`, amelyben a függőségeket sorolja fel. Ezeket a függőségeket az első letöltésük után elmenti, hogy későbbi használat esetén lokálisan, a számítógép tárhelyéről elérhesse azokat. A *package.json* leíró mellett megtalálható még egy *package-lock.json* állomány, amely a függőségek telepítésekor generálódik. Ez a függőségek verzióinak pontos nyomonkövetését segíti, valamint a tranzitív függőségeket tartalmazza.

3.2. Web technológiák

Angular

Az Angular [42] egyoldalas web applikációk létrehozására alkalmas nyílt forráskódú keretrendszer, amelyet a Google fejlesztett ki. Az egyoldalas webalkalmazások előnye az, hogy a kommunikáció során nem szükséges a teljes oldal újratöltődjön, hanem dinamikusan csak egyes elemei változnak. A szükséges forráskódok (HTML, CSS és TypeScript) csak az oldal megnyitásakor töltődnek be, majd az alkamazásszervertől kapott adatok dinamikusan jelennek meg.

Programozási nyelve a TypeScript, habár létezik JavaScript alapú verziója is (AngularJS). A TypeScript kódot nem tudják értelmezni a böngészők, ezért futás előtt lefordítja azt JavaScript kódra.

A felhasználók számára megjelenő nézetet komponensek építik fel, azok felelnek az üzleti logika elvégzéséért. Minden komponens tartalmaz egy sablont, amelyik a kinézetet határozza meg, valamint a vezérlést végző utasításokat. A sablonok és a vezérlés közötti összeköttetést az Angular kötések segítségével végzi el (data-binding). Az Angular-nek megvan a saját DI (Dependency Injection) [41] keretrendszere , amely elősegíti az alkalmazások hatékonyságát és modularitását.

Material Design

A Material Design [32] egy formanyelv, amely fejlesztők számára biztosít iránymutatásokat, komponenseket, illetve eszközöket, amelyek támogatják egy felhasználóbarát felület létrehozását. Lehetővé teszi, hogy többféle platformon is hasonló megjelenítést adjon az applikációknak.

Az *Angular Material* [13] egy komponens könyvtár, amely ezeket az elveket valósítja meg a saját komponensein keresztül. Egységes kinézetet biztosít a különféle komponensek között.

Sass

A Sass [47] egy stíluslap nyelvezet, amely a CSS nyelv egy kiterjesztése. A Sass lehetővé teszi, hogy változókat, függvényeket, beágyazásokat és modulokat használjunk, ezzel elősegítve a kód átláthatóságát és hozzáférhetőségét. Teljesen CSS kompatibilis kódot eredményez, amely könnyedén lefordítható. Az Angular Material használja egyedi témák létrehozásánál.

Responsive Design

A Responsive API [5] interfész biztosítja azt, hogy a komponensek mérete igazodjon a képernyő méretéhez. CSS MediaQuery [45] definiciókhoz rendel különböző álneveket, amelyek használatával beállíthatóak a komponensek specifikus tulajdonságai az adott képernyő méretéhez viszonyítva.

A Flex Layout [12] egy elrendezési API, amely biztosítja a komponensek elrendezését egy adott konténeren belül. A Responsive API motor lehetővé teszi, hogy HTML kódból könnyedén méretre igazíthatóak legyenek a komponensek. Segítségével megszabható az elrendezés, méretezés vagy az elemek láthatósága.

3.3. Android technológiák

Android

Az Android [31] egy mobil operációs rendszer, amely a Linux kernelre [4] épül és érintőképernyős eszközökre lett főként kifejlesztve. A jelenlegi legújabb stabil verziója az Android 10, amelyet 2019 szeptemerében adtak ki.

A Linux kernelt minden Android operációs rendszerrel rendelkező mobiltelefont készítő cég személyre szabhat a saját készülékeihez. A kernel által biztosított funkciókat különböző programkönyvtárak használják, mint például az SQLite. Ezek a könyvtárak C/C++ nyelvekben vannak implementálva. A következő szinten az Android futókörnyezet van, a Dalvik Virtual Machine, amely a JVM-hez (Java Virtual Machine) hasonlít, viszont kifejezetten Android-ra

van optimalizálva. Ez a szint végzi a memóriakezelést és többszálusítást is. A futókörnyezet fölött található egy keretrendszer, amely magas szintű erőforrásokat biztosít értesítések, felhasználói felületek kezelésére. A legfelső szinten az alapvető Android alkalmazások találhatóak meg: névjegyzék, játékok, stb.

Kotlin

A Kotlin [24] egy platformfüggetlen programozási nyelv, melyet a JetBrains fejlesztett ki, hogy egy olyan programozási nyelvet hozzanak létre, amiben megtalálhatóak a többi nyelvből hiányzó funkciók (*Null safety*, *Data class*, *Co-routines*). Bejelentésére 2011-ben került sor, 2012-ben pedig kiadták az első nyílt forráskódú verziót. Az első stabil verzió (v1.0) 2016 feruárjában jelent meg. Egy évvel ezután, 2017-ben a Google bejelentette, hogy a Java mellett a Kotlin-t is elfogadja, mint Android fejlesztői nyelv, majd további fejlesztések után 2019 májusában bejelentették, hogy a Kotlin az előnyben részesített nyelv.

Gradle

A Gradle [22] egy nyílt forráskódú, autómatizált build folyamatok kezelésére létrehozott eszköz, amely Android oldalon került alkalmazásra. Gradle esetén megtalálható egy központi, a telepítést leíró állomány, a `build.gradle`, amely az alkalmazás gyökerében található. A leíró állomány `Groovy` [14] programozási nyelvben van definiálva, de lehetőség adódik más programozási nyelveket is használni. Itt lehet `build` feladatokat definiálni, amelyek parancssorból futtathatóak. A feladatok mellett itt adható meg, hogy milyen függőségekre van szüksége a projektnek. Ezeket a függőségeket egy központi tarolóból tölti be, majd első letöltés után lokálisan el is menti, így későbbi használat esetén nem kell újra letöltse, csupán használja azt.

Material design

A web klienshez hasonlóan (lásd 3.2. fejezet), a design kialakítására az Android alkalmazás is Material Design-t [32] alkalmaz.

3.4. iOS technológiák

IOS

Az iOS az Apple saját mobil operációs rendszere, kizárólag az Apple telefonkészülékei számára fejlesztik. Az első SDK megjelenése után vette át jelenlegi nevét, korábban az iPhone OS nevet viselte, amit Steve Jobs adott neki. Jelenlegi, legújabb verziója 2020. január 28.

óta a 13.3.1-es. A második legnépszerűbb mobil operációs rendszer az Android után, és nagyjából 178 millió alkalmazása található meg az `Apple Store`-on. A legtöbb alkalmazás `Objective-C`-ben íródott, de a 2014-ben megjelent `Swift` rövid időn belül felváltotta azt. Az architektúrája XNU Kernel kernel alapú. Az Apple a számítástechnika egyik meghatározó úttörője, amely leginkább a grafikus felület kialakításával hívja fel magára a figyelmet.

Swift

Az iOS kliensalkalmazás Swift programozási nyelvben íródott, amelyet szintén az Apple fejleszt. A Swift egy általános célú, multiparadigmás programozási nyelv, amely úgy a biztonság, mint a kényelmes fejlesztés és jó futásidejű teljesítmény terén a programozás modernizálására törekszik. Nagyon könnyen tanulható nyelv, az iOS mellett macOS, watchOS, és tvOS platformokra is lehet vele fejleszteni. A programozási nyelv elősegíti a fejlesztőt az olyan módszerek, minták használatát, amelyek segítségével a kód biztonságosabb lesz. A legújabb verzió 5.1-es és nyílt forráskódú.

CocoaPods

A CocoaPods [1] egy csomagkezelő rendszer, amely Objective-C és Swift függőségek telepítésére és karbantartására használható. Az alkalmazás fejlesztése során több ilyen külső könyvtár is felhasználára került.

3.5. Eszközök

Docker

A Docker [33] egy konténer-alapú virtualizációs eszköz, amely arra ad lehetőséget, hogy az alkalmazások egy elszigetelt, virtuális környezetben fussank. Docker használata megóvja az alkalmazást olyan külső tényezöktől, amelyek negatívan befolyásolhatják, ilyen például az operációs rendszerek közti eltérések. Előnye más virtualizációs eszközökkel szemben az, hogy könnyű kezelni illetve kevés helyet foglalnak, ezáltal lényegesen gyorsabb.

Docker-compose

A Docker compose [2] a Docker konténerek kezelésére alkalmas eszköz. Ennek segítségével valósítható meg az, hogy az szerver és a hozzá tartozó adatbázis külön konténerben fussanak, így az alkalmazás különálló moduljai könnyebben elkülöníthetőek. Docker compose segítségével az alkalmazás skálázhatósága is sokkal könnyebbé válik.

Git

A Git [25] az egyik legnépszerűbb verziókövető rendszer. A Git tárolón az alkalmazás adott időben elmentett állapotai vannak eltárolva, így nyomon követhetőek a változások. A projekt esetén öt tároló van létrehozva, a `diaknapp-api-server`, `diaknapp-web-client`, `diaknapp-ios-client` és `diaknapp-andoird-client`, illetve a kitelepítés érdekében egy ötödik tároló, a `diaknapp-compose`.

GitLab

A *GitLab* [46] egy `DevOps` platform, amelynek segítségével egyszerűen nyomon követhető a Git tárolón történő műveletek. Egy hasznos és egyszerű felhasználói felületet biztosít, ahol nem csak a változásokat lehet ellenőrízni, de a projektbe történő integráció is nyomonkövethető. Ezáltal egy csapat tagjai könnyedén tudják ellenőrizni társaik kódját. Továbbá a Gitlab egy táblát is biztosít, amely a projektekkel kapcsolatos tennivalók felvezetésére alkalmas.

A GitLab tároló segítségével lett megvalósítva a kitelepítés is. Gitlab esetén lehetőség van `pipeline`-okat futtatni, amely segítségével statikus kódellenőrzés és build-folyamatok vezérelhetőek. Mikor a definiált `pipeline`-ok sikeresen lefutnak, a GitLab létrehoz egy `docker image`-t a projekt aktuális állapotából és elmenti azt a `GitLab registry`-be. A `diaknapp-compose` tároló összegyűjti ezeket az állapotokat (web és szerver), majd a megadott címre kitelepíti azokat. A `diaknapp-compose` tároló küllönlegessége még az, hogy egy NGINX [43] webszervert tartalmaz, amely fogadja a bejövő kéréseket majd az elérési útvonal alapján továbbítja azt a szervernek vagy a webalkalmazásnak. Azon kérések, amelyek a `/api` címre érkeznek a szerverhez irányítja, a többit pedig a web kliensnek.

GitKraken

A *GitKraken* [3] egy fejlesztőknek szánt grafikus felhasználói felület, amelyet a Git használatának leegyszerüsítésére fejlesztettek ki. A program segítségével a felhasználók gyorsan és könnyedén tudnak új változásokat bevinni a rendszerbe, valamint váltani a már elmentett verziók között.

MySQL Workbench

A MySQL Workbench [34] egy grafikus felhasználói felület, amelyet a MySQL adatbázis könnyed használatának érdekében fejlesztettek ki. Ennek segítségével könnyedén lehet ellenőrizni az adatbázisban létrejött táblákat, illetve különböző műveletek végezhetőek el rajtuk.

Visual Studio Code

A Visual Studio Code [9] egy pehelysúlyú kódszerkesztő, amelyet a Microsoft fejlesztett ki és bármely operációs rendszeren futtatható. Nagy előnye a többi kódszerkesztőhöz képest, hogy könnyedén hozzáadhatóak bővítmények. A Visual Studio Code a web kliens és a szerver fejlesztésénél került használatra.

Android Studio

Az Android Studio [48] Android alkalmazások fejlesztésére alkalmas fejlesztői környezet, amely bármely operációs rendszeren használható. A Visual Studio Code-hoz hasonlóan itt is megtalálható a Git verziókövető támogatás. Biztosít továbbá egy emulátort, amely segítségével alkalmazásainkat különböző méretű és verziójú Android operációs rendszerekkel ellátott készülékeken tudjuk tesztelni.

Xcode

Az iOS mobilalkalmazás az Apple által fejlesztett Xcode [10] kódszerkesztővel került megvalósításra. Az Xcode környezeten belül lehetőségünk van futtatni különböző iOS mobileszköz szimulátorokat. Az Interface Builder segítségével könnyedén modósíthatóak a felhasználói felületek, komponensek.

4. A kliensalkalmazások működése

Ebben a fejezetben a három kliensalkalmazás funkcionalitásainak használata kerül bemutatásra. A működést rövid leírás és néhány képernyőfotó szemlélteti. A két mobilalkalmazás kinézete hasonló, ezért azok egy alfejezetben vannak bemutatva.

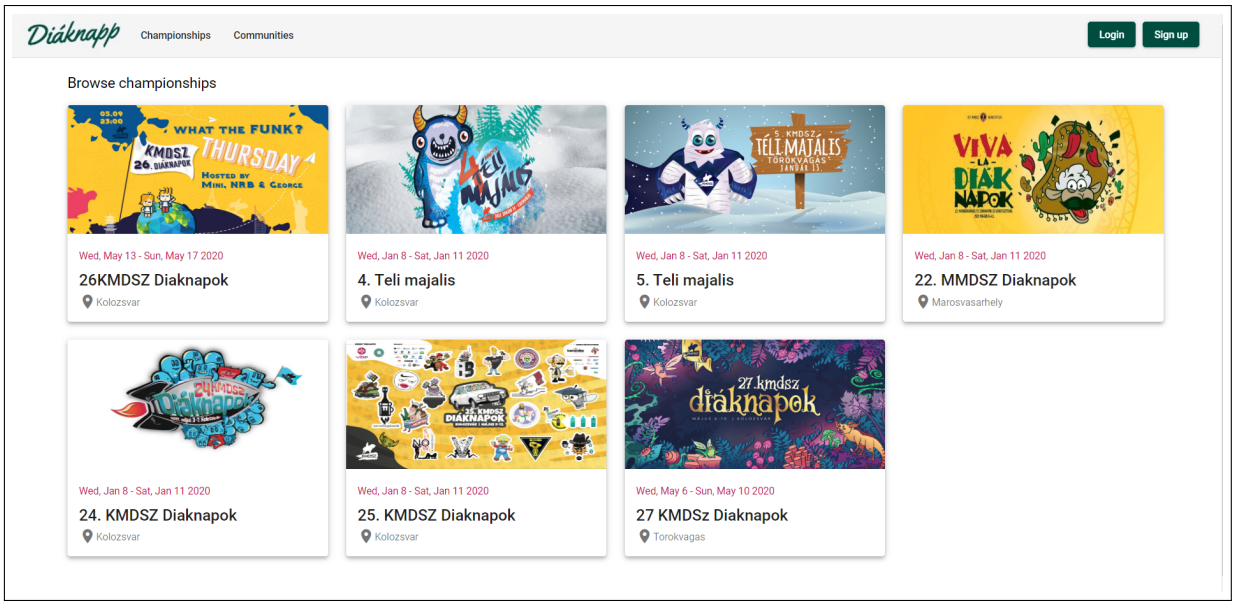
4.1. A webes felület használata

A weboldal megnyitásakor a bajnokságok listája nyílik meg, ahol néhány fontos információ jelenik meg minden bajnokságról, amelyeket a 8. ábra szemléltet.

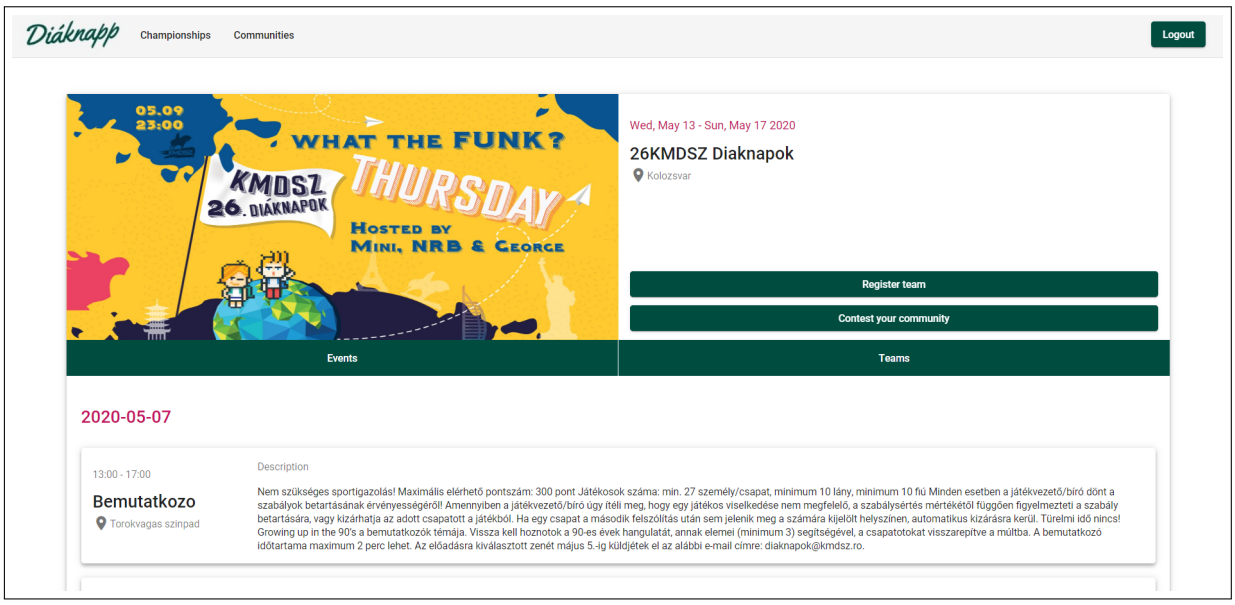
Az oldal bal felső sarkában, a logó mellett található két navigációs gomb, amelyek segítségével az alkalmazás két fő oldala között lehet váltani: a bajnokságok listája (*Championships*) és a közösségek listája (*Communities*). A bajnokságokhoz hasonlóan a közösségekről is látható néhány fontosabb információ.

A jobb felső sarokban levő gombokkal be lehet jelentkezni a weboldalra, illetve lehet regisztrálni. Ezek gombok valamelyikét megnyomva előugrik egy dialógus ablak, amelyben a felhasználó megadhatja az adatait, amellyekkel regisztrálni vagy bejelentkezni szeretne.

A bajnokságok listájában látható kártyák közül az egyiket kiválasztva a bajnokság oldala lesz látható, ahol részletesebb információk jelennek meg (lásd 9. ábra). Alapértelmezetten mutatja a hozzá tartozó események listáját, csoportosítva azokat aszerint, hogy milyen napra esnek. A bajnokság oldalán található *Events* és *Teams* gombok segítségével lehet nézetet váltani, vagyis a *Teams* gombra kattintva az események listája helyett a csapatok listája lesz látható. Ez a két gomb egy belső navigációként működik, vagyis egyszerre mindig csak az egyiket



8. ábra. Az oldal megnyitásakor a bajnokságok listája fogad



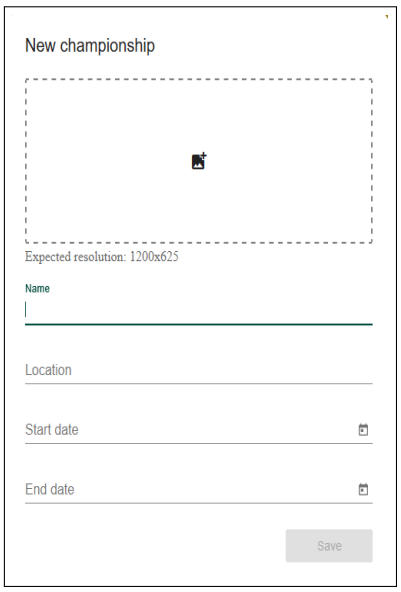
9. ábra. Részletes információk a kiválasztott bajnokságról, valamint a hozzá tartozó események listája

láthatjuk: az események listáját vagy a csapatokét. Egy szervezőnek ezen kívül a csapatok listájában megjelenik az is, hogy a csapat jelentkezését már elfogadták vagy sem. A csapatok mellet megjelenik egy szerkesztés gomb, amely előhoz egy dialógust, amely mutatja a csapat adatait, valamint a szervezőnek lehetőséget ad, hogy elfogadja vagy elutasítsa a jelentkezését a csapatnak.

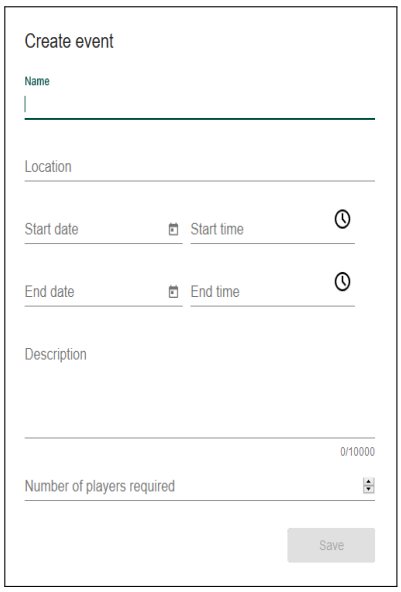
A szervezők a bajnokság oldalán tudják szerkeszteni az adott bajnokság információit, a jobb felső sarokban levő gombra kattintva. Ezzel a gombbal előjön egy dialógus, amely hasonlít a 10. ábrán látható dialógushoz, azzal a különbséggel, hogy az adott bajnokság adataival ki van töltve minden mező. Így a szervezők tudják módosítani a bajnokság részleteit.

A 9. ábrán látható még két másik gomb is: *Register team* és *Contest your community*. Ezek a gombok csak akkor jelennek meg, hogyha a felhasználó be van jelentkezve. A *Register team* gombra kattintva előugrik egy dialógus, amely segítségével egy csapatot lehet létrehozni. Itt a felhasználó meg kell adja a csapat adatait, valamint más felhasználók e-mail címének a megadásával adhat hozzá csapattársakat a csapatához, és hogyha sikerül regisztrálnia a csapatot, akkor annak ő lesz a csapatkapitánya. A szükséges adatokat a 12. ábra szemlélteti. Abban az esetben, ha felhasználó egy közösségnek a vezetője, akkor regisztrálhat egy csapatot a *Contest your community* gombra kattintva, amely előhoz számára egy - a csapatok regisztrációjához hasonló - dialógust, azzal a különbséggel, hogy nem szükséges minden csapattársa e-mail címét beírnia, hanem kiválaszthatja a közösséget egy legördülő listából. Amiután kiválasztotta a közösséget, a tagok megjelennek egy listában amiből csak ki kell jelölnie a megfelelő tagokat.

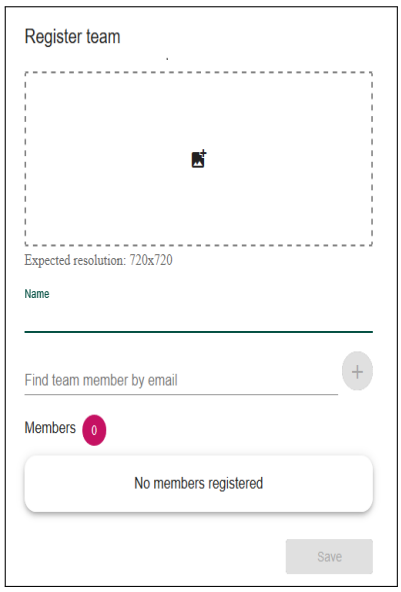
A főoldal jobb alsó sarkában megjelenik egy plusz gomb, azon felhasználóknak akik épp be vannak jelentkezve. Ezzel a plusz gombbal megnyílik egy dialógus, amellyel új



10. ábra. Új bajnokság létrehozása



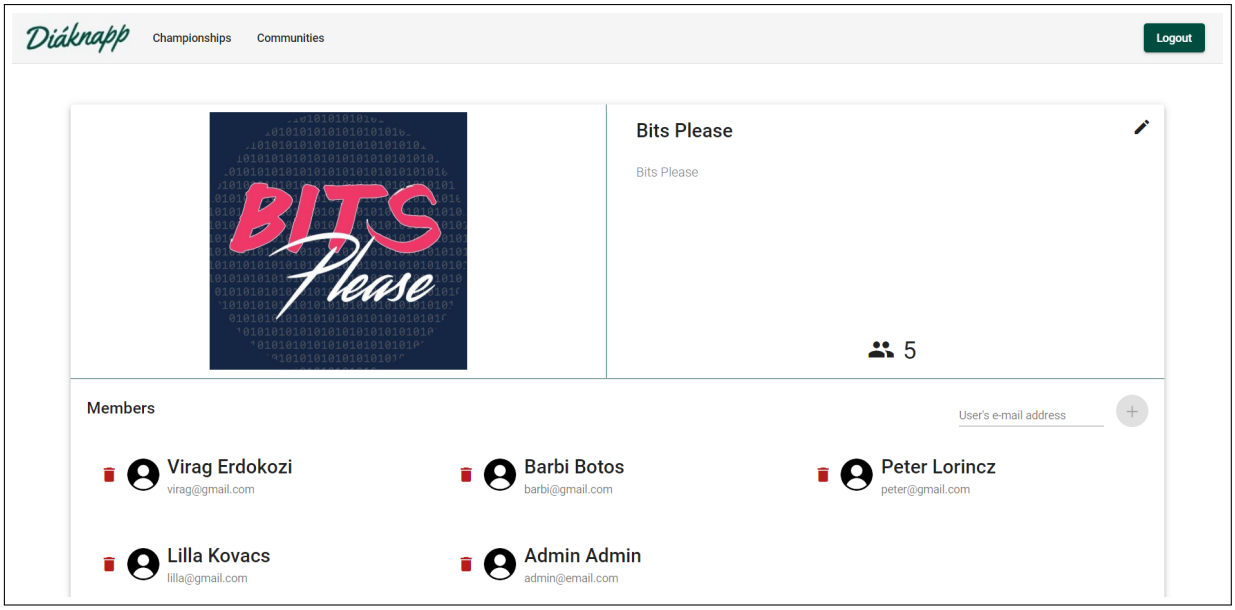
11. ábra. Új esemény létrehozása



12. ábra. Új csapat regisztrálása

bajnokságot lehet létrehozni (lásd 10. ábra). Hogyha sikerül létrehozni az adott bajnokságot, akkor automatikusan megjelenik a weboldalon az új bajnokságnak az oldala. Ezen az oldalon a jobb alsó sarokban szintén megjelenik egy plusz gomb, viszont ennek a gombnak más a funkciója. Ez a gomb csak az a felhasználó számára elérhető, aki létrehozta a bajnokságot, vagyis szervező szerepkörrel rendelkezik. A gomb megnyomása által megnyílik egy dialógus, amellyel a bajnoksághoz egy új eseményt lehet létrehozni (lásd 11. ábra).

Ahogyan a fejezet elején is említve van, a bal felső sarokban levő *Communities* gombra rákattintva megjelenik a közösségek listája. A bejelentkezett felhasználóknak megjelenik a jobb alsó sarokban egy plusz gomb, amelyre rákattintva megjelenik a dialógus, amely segítségével



13. ábra. A közösségvezető számára megjelenő információk a közösségről

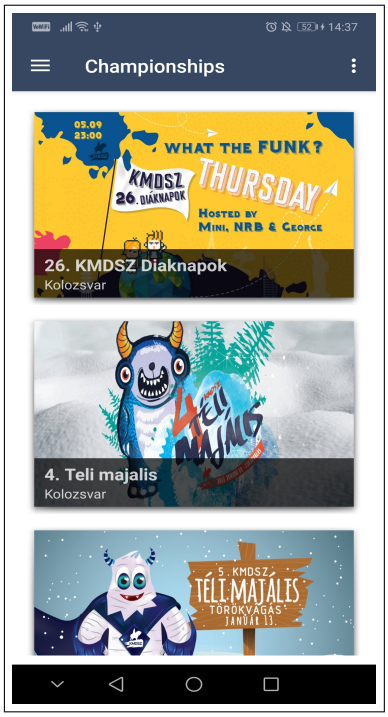
új közösséget lehet létrehozni. Az itt látható kártyák közül ki lehet választani az egyiket, ami átvisz a kiválasztott közösség oldalára, amely a 13. ábrán látható. A közösségvezető számára megjelennek a közösség tagjainak listája is, viszont ezt a többi felhasználó nem láthatja, amikor az oldalra navigál. A közösség részletes információit tartalmazó kártya jobb felső sarkában található a szerkesztés gomb, amelyre rákattintva előjön a dialógus, amellyel létre hozhatunk új közösséget, azzal a különbséggel, hogy az adatok ki vannak töltve a közösségről. Ugyanezen a dialóguson jelenik meg a törlés gomb is.

A közösségvezető hozzá tud adni felhasználókat a közösséghez az e-mail címük megadásával. A közösség kártyáján jobb oldalt megjelenik egy form, ahol megadhatja a közösségvezető az e-mail címet és a plusz gombra kattintva elküldi a kérést. Hogyha létezik az a felhasználó, akkor megjelenik a neve a csapattagok listájában.

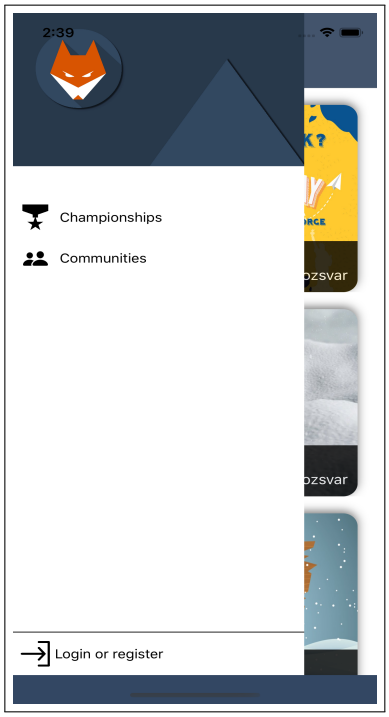
4.2. A mobilos felületek használata

A telefonos alkalmazások megnyításakor a webes felülethez hasonlóan a bajnokságok listája jelenik meg, melyet a 14. ábra szemléltet. Itt minden bajokságnak megjelenik a képe, helye, illetve a kezdő és végső dátuma, ha ugyanazon a napon kezdődik és jár le, akkor azt egy dátumként jeleníti meg.

A bal felső sarokban egy menü ikon található. Ezt kiválasztva egy oldalsó navigációs menü jelenik meg, ami a képernyő bal oldaláról úszik be az oldal fölé. Ez jól látható s 15. ábrán.

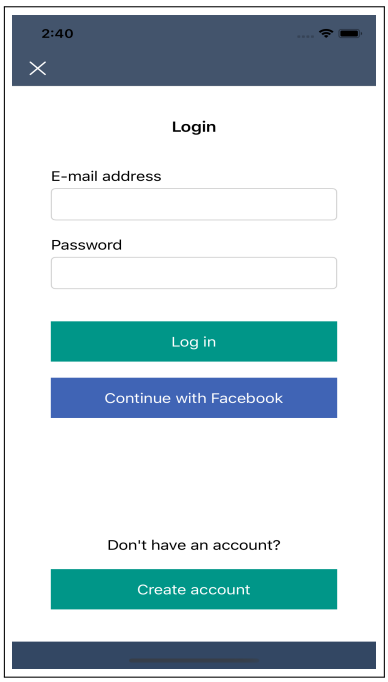


14. ábra. Bajokságok listázása - Android

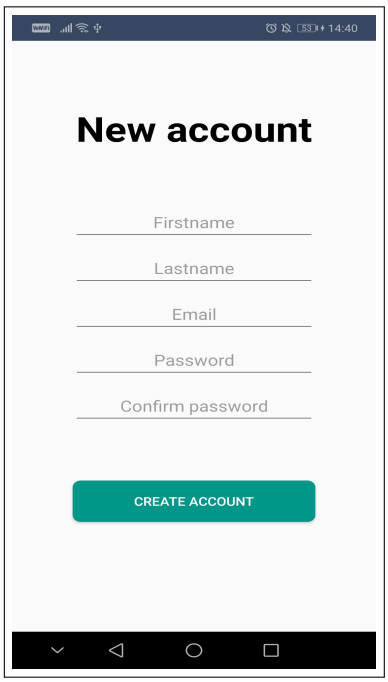


15. ábra. Navigációs menü - iOS

16. ábra. Közösségek listázása - Android



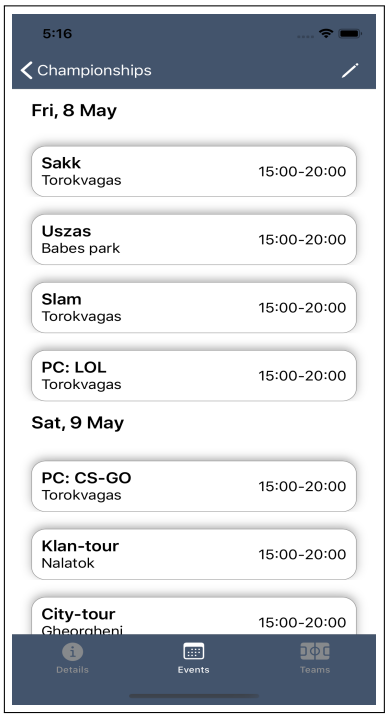
17. ábra. Bejelentkezés - iOS



18. ábra. Regisztráció - Android

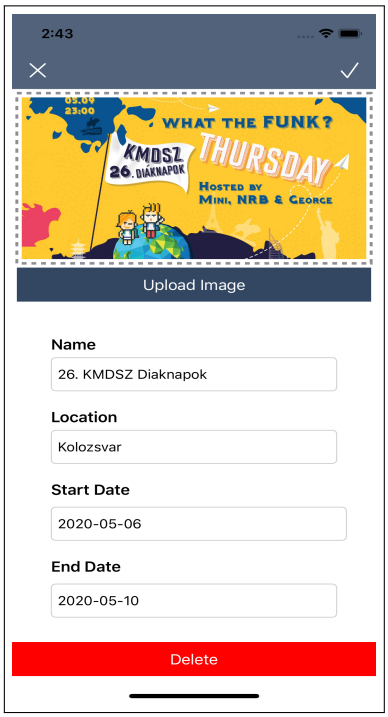
Amennyiben a vissza gomb kerül lenyomásra vagy a navigációs menün kívüli részre kattintunk akkor eltűnik a menü és az előzőleg megnyitott oldal ismét láthatóvá válik.

A navigációs menü `Communities` gombját kiválasztva, megjelenik egy új oldal, amely a közösségeket listázza. Ezen az oldalon megjelennek a közösségekhez feltöltött képek valamint



19. ábra. Események listázása - iOS

20. ábra. Esemény részletei - Android



21. ábra. Bajnokság szerekesztése - iOS

maga a közösség neve látható. Ez az oldalt a 16. ábrán tekinthető meg.

Amennyiben a menüben a `Login or register` gombra kattint a felhasználó a 17. ábrával szemléltetett bejelentkezést lehetővé tevő oldal jelenik meg. Innen a `Register account` gombal tud tovább navigálni a regisztrációs oldalra, amelyet a 18. ábra szemléltet.

Abban az esetben, ha bajnokságok oldalán (14. ábra) a felhasználó kiválasztja valamelyik bajnokságot, akkor megjelenik egy új oldal (19. ábra), amely a hozzá tartozó eseményeket listázza. Itt böngészhető néhány információ az eseményekről: esemény neve, helyszíne illetve megrendezésének időpontja. Ha az események oldalán a felhasználó kiválasztja valamelyik eseményt részletesebb léírást kaphat róla, amit a 20. ábra mutat.

A felhasználó módosítani tudja a bajnokságot, abban az esetben, ha be van jelentkezve, és ő az, aki létrehozta azt. Ezt úgy teheti meg, ha kiválaszt egy bajnokságot és a megjelenő képernyő jobb felső sarkában található módosítás gombot választja ki. Ez eredményezi a bajnokság adatainak megváltoztatását biztosító oldal megjelenését, amelyet a 21. ábra szemléltet.

Következtetések és továbbfejlesztési lehetőségek

A DiáknApp projekt fejlesztése során létrejött egy szoftverrendszer, amely biztosít egy webes, illetve natív Android és iOS alkalmazásokat is, amely a diáknapokhoz hasonló bajnokságok szervezését könnyíti meg. A diákszövetségek nyilván tudják tartani az eseményeket, a résztvevők pedig nyomon tudják követni azokat.

A fejlesztés első szakaszában különböző ötletek fogalmazódtak meg a leendő szoftver funkcionalitásairól, amelyek egy részét sikerült megvalósítani. A maradt ötletek adják a rendszer továbbfejlesztési lehetőségeit, amelyek között szerepelnek például:

- Az alkalmazás többnyelvűsítése, mivel jelenleg a kulcsszavak mindegyike angol nyelven látható, valamint a dátumok egységesítése;
- Annak a szervezőnek aki létrehozta a bajnokságot jogot adni arra, hogy más felhasználóknak is tudjon szervezői jogot adni;
- Hogyha már van több szervező is egy bajnokságon akkor egyes szervezőknek bírói jogot lehet adni, aki egy adott eseményen pontozhatja a csapatok teljesítményét, majd azokat egy oldalon be tudja vezetni a rendszerbe;
- A csapatok ranglistájának megtekintését az adott bajnokságon, és szűrési lehetőség megadása, hogy a pontozást meg lehessen tekinteni az adott események pontozása szerint (például a kosárlabda mérkőzések pontjai alapján szűrni a csapatokat);
- Mobil platformokon értesítési rendszer (push notifications) bevezetése, hogy egy csapattag kapjon értesítést, amikor egy esemény elkezdődik;
- Egy beosztási rendszer bevezetése, ahol a csapattagok tudnak jelentkezni az eseményekre, amelyeken részt szeretnének venni, majd a csapatkapitány jóváhagyásával megjelenik a beosztás a szervezőknek is. A csapatkapitánynak meg lehetne adni a lehetőséget, hogy módosítsa ezt a listát vagy esetleg hozzáadjon csapattagokat, akik még nem jelentkeztek más eseményekre;
- Térkép nézet bevezetése, amelyen megtekinthetőek lennének az események;
- Chat rendszer bevezetése, a könnyű és direkt kommunikáció megkönnyítése érdekében.

A projekt alapötlete a kolozsvári diáknapok megszervezésének a nehézsége volt, valamint az hogy a szervezők és a résztvevők közötti kommunikáció legtöbbször a Facebook platformon történik. Ezeknek a nehézségeknek a kiküszöbölésére jött létre a DiáknApp alkalmazás, amely megkönnyíti mind a szervezők, mind a csapatok dolgát a bajnokságok lebonyolításában.

Hivatkozások

[1] *A CocoaPods hivatalos weboldala*. URL: https://cocoapods.org/ (elérés dátuma: 2020. máj. 1.)

[2] *A Docker compose hivatalos weboldala*. URL: https://docs.docker.com/compose/ (elérés dátuma: 2020. ápr. 18.)

[3] *A GitKraken hivatalos weboldala*. URL: https://www.gitkraken.com (elérés dátuma: 2020. ápr. 18.)

[4] *A Linux kernel hovatalos weboldala*. URL: https://www.kernel.org/doc/ (elérés dátuma: 2020. ápr. 18.)

[5] *A Responsive API hivatalos weboldala*. URL: https://github.com/angular/flex-layout/wiki/Responsive-API (elérés dátuma: 2020. máj. 2.)

[6] *A Retrofit hivatalos weboldala*. URL: https://square.github.io/retrofit/ (elérés dátuma: 2020. ápr. 28.)

[7] *A Sequelize hivatalos weboldala*. URL: https://sequelize.org/v5/ (elérés dátuma: 2020. márc. 13.)

[8] *A V8 hivatalos weboldala*. URL: https://v8.dev/docs (elérés dátuma: 2020. ápr. 27.)

[9] *A Visual Studio Code hivatalos weboldala*. URL: https://code.visualstudio.com/ (elérés dátuma: 2020. ápr. 18.)

[10] F. Anderson. *Step Into Xcode: Mac OS X Development*. Addison-Wesley, 2006.

[11] *Az Alamofire hivatalos weboldala*. URL: https://github.com/Alamofire/Alamofire (elérés dátuma: 2020. ápr. 24.)

[12] *Az Angular Flex-Layout hivatalos weboldala*. URL: https://github.com/angular/flex-layout/wiki (elérés dátuma: 2020. máj. 3.)

[13] *Az Angular Material hivatalos weboldala*. URL: https://material.angular.io/ (elérés dátuma: 2020. ápr. 17.)

[14] K. Barclay és J. Savage. *Groovy Programming: An Introduction for Java Developers*. Elsevier Science, 2010.

[15] Ryan Boyd. *Getting started with OAuth 2.0*. " O'Reilly Media, Inc.", 2012.

[16] P. DuBois. *MySQL*. Pearson Education, 2008.

[17] Patrycja Dybka. *ORMs Under the Hood*. URL: https://www.vertabelo.com/blog/orms-under-the-hood/ (elérés dátuma: 2020. máj. 3.)

[18] *Fetch API hivatalos weboldala*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (elérés dátuma: 2020. ápr. 19.)

[19] Wayne Graham. *Facebook API Developers Guide*. firstPress, 2008.

[20] M. Haverbeke. *Eloquent JavaScript: A Modern Introduction to Programming*. No Starch Press, 2011.

[21] Tankó Helén. *26. KMDSz Diáknapok*. URL: https://kmdsz.ro/KMDSZ/aktualis?id=92 (elérés dátuma: 2020. ápr. 25.)

[22] K. Kousen. *Gradle Recipes for Android: Master the New Build System for Android*. O'Reilly Media, 2016.

[23] Suresh Kumar. *How JSON Web Token(JWT) authentication works?* URL: https://medium.com/@sureshdsk/how-json-web-token-jwt-authentication-works-585c4f076033 (elérés dátuma: 2020. máj. 3.)

[24] A. Kurniawan. *Practical Kotlin Programming*. PE Press.

[25] J. Loeliger és M. McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media, 2012.

[26] L.A. MacVittie. *XAML in a Nutshell*. O'Reilly, 2006.

[27] D. Maharry. *TypeScript Revealed*. Apress, 2013.

[28] J. Manning, P. Buttfield-Addison és T. Nugent. *Learning Swift: Building Apps for macOS, iOS, and Beyond*. O'Reilly Media, 2018.

[29] R.C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2018.

[30] M. Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, 2011.

[31] Z. Mednieks és mások. *Programming Android*. O'Reilly Media, Incorporated, 2011.

[32] Kyle Mew. *Learning Material Design*. Packt Publishing Ltd, 2015.

[33] A. Mouat. *Using Docker: Developing and Deploying Software with Containers*. O'Reilly Media, 2015.

[34] *MySQL Workbench hivatalos weboldala*. URL: https://dev.mysql.com/doc/workbench/en/ (elérés dátuma: 2020. ápr. 18.)

[35] T. Nurkiewicz és B. Christensen. *Reactive Programming with RxJava: Creating Asynchronous, Event-Based Applications*. O'Reilly Media, 2016.

[36] D. Parker. *JavaScript with Promises: Managing Asynchronous Code*. O'Reilly Media, 2015.

[37] S.K. Patel. *Instant Gson*. Packt Publishing, 2013.

[38] S. Powers és an O'Reilly Media Company Safari. *Installing, Maintaining, and Publishing Node Modules with Npm*. O'Reilly Media, Incorporated, 2016.

[39] J. Russell és R. Cohn. *Model View Viewmodel*. Book on Demand, 2012.

[40] E. Sadun és an O'Reilly Media Company Safari. *IOS Auto Layout Demystified, Second Edition*. Addison-Wesley Professional, 2013.

[41] V. Savkin és J. Cross. *Essential Angular*. Packt Publishing, 2017. 5. fejezet.

[42] S. Seshadri. *Angular: Up and Running: Learning Angular, Step by Step*. O'Reilly Media, 2018.

[43] R. Soni. *Nginx: From Beginner to Pro*. Apress, 2016.

[44] B. Syed. *Beginning Node.js*. Apress, 2014.

[45] *Using media queries*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries (elérés dátuma: 2020. ápr. 28.)

[46] Jeroen Van Baarsen. *GitLab Cookbook*. Packt Publishing Ltd, 2014.

[47] L. Watts. *Mastering Sass*. Packt Publishing, 2016.

[48] M. Wolfson és D. Felker. *Android Developer Tools Essentials: Android Studio to Zipalign*. O'Reilly Media, 2013.