XXI. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)
Kolozsvár, 2018. május 24–27.

Daily Challenge

Kihívások kezelése mobil alkalmazással

Szerzők:

Gagyi Mátyás
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év
Patakfalvi Örs Krisztián
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:

Sulyok Csaba, doktorandusz,
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar
Ráduly Sándor, szoftverfejlesztő,
Codespring

Kivonat

Napjainkban egyre népszerűbbek a különféle kihívások, amelyekben a résztvevők bizonyos időnként meghatározott lépéseket kell elvégezzenek, például hetente egy könyv kiolvasása vagy harminc napig a has edzése. Noha a lelkesedők szívesen részt vesznek, hiányt szenvednek egy egységes és átlátható platformban.

Az Daily Challenge projekt célja, hogy egy helyen elérhetővé tegyen minél több kihívást, egységes felületet nyújtson a felhasználók számára és megkönnyítse a részvételt.

A projekt három komponensre osztható. A központi szerver egy skálázható platformot emel az adatbázisunk köré és elérési opciót biztosít tetszőleges számú kliensnek. A több platformra kompilált natív mobilalkalmazáson és a webes felületen keresztül a különböző szerepkörű felhasználók követhetik és karbantarthatják a kihívásokat.

A dolgozat áttekintést nyújt a rendszer működéséről, az architektúrájáról, részletezi a funkcionalitásokat, a felhasznált technológiákat, eszközöket, valamint szemlélteti az alkalmazás működését.

Tartalomjegyzék

Bevezető

Napjainkban egyre népszerűbbek a különféle kihívások, amelyekben a résztvevők bizonyos időnként meghatározott lépéseket kell elvégezzenek, például harminc napig a különböző testrészek edzése vagy hetente egy könyv kiolvasása, esetleg egy különleges fénykép elkészítése (1. ábra[1]). Noha a lelkesedők szívesen részt vesznek, hiányt szenvednek egy egységes és átlátható platformban, ahol nyomonkövethetik a teljesítményüket, véleményeket olvashatnak a kihívásokról vagy akár barátaikat is ösztönözhetik.

A Daily Challenge alkalmazás egy kihívásokon alapúló szociális platform szerepét hivatott betölteni a mindennapjainkban. A projekt elsődleges célja, hogy egy helyen elérhetővé tegyen minél több kihívást, egységes felületet nyújtson a felhasználók számára és megkönnyítse a részvételt.

A projekt három fő komponensre osztható. A központi szerver egy skálázható platformot emel az adatbázisunk köré és elérési opciót biztosít tetszőleges számú kliensnek. A natív mobilalkalmazás, amely elérhető iOS és Android eszközökön, lehetőséget biztosít a felhasználóknak, hogy új kihívásokat találjanak maguknak és azokat a részletes leírások segítségével egy kellemes időtöltés keretében elvégezhessék. A webalkalmazás a tartalommenedzserek számára nyújt betekintést a felhasználók által ajánlott kihívás-ötletekbe, karbantarthatják a már elkészült kihívásokat. Az adminisztrátorok szintén a webes felület segítségével felügyelhetik a felhasználók tevékenységét és jogköreit.

1. ábra. Az interneten rengeteg hasonló kihívás terjed, ám sehol nincsenek összegyűjtve.

[1] Ábrák forrásai: https://ro.pinterest.com/pin/26177241559072541/ és https://www.pinterest.co.uk/pin/534661787006074924/, utolsó megtekintés dátuma: 2018-04-16

A dolgozat 1. fejezetében az elkészült funkcionalitások kerülnek bemutatásra, felhasználói jogkörökre lebontva. Kifejtésre kerülnek az általános felhasználó lehetőségei a mobil applikáció keretein belül, valamint a tartalommenedzserek és az adminisztrátorok jogosultságai a webes felületen. A 2. fejezetben feltárjuk a projekt architektúráját, lebontva szerver, mobil és web részre. Az ezt követő 3. fejezet tartalmazza az általunk a fejlesztési folyamat alatt használt technológiákat és eszközöket. A 4. fejezet részletesen bemutatja a moibil alkalmazás és a webes felület működését, végigvezérel minket a felhasználók elé táruló lehetőségeken, ábrákkal szemléltetve mindezt. Az 5., egyben utolsó fejezet tartalmazza következtetéseinket és a Daily Challenge projekt jövőjére vonatkozó terveinket.

A projekt saját ötletből fejlődött ki a Codespring mentorainak segítségével. A fejlesztés kezdeti szakaszában, az egyetem keretein belül csatlakozott hozzánk egy félév ereéig három évfolyamtársunk, Rácz Orsolya, Kún Szabolcs-Hunor és Vass Tekla.

Ezúton szeretnénk megköszönni diáktársainknak a hozzájárulásukat, Simon Károlynak a mentorprogram létrehozását, valamint mentorainknak, Sulyok Csabának és Ráduly Sándornak, a szakmai támogatást és segítséget.

1. Funkcionalitások

A projekt legfőbb funkcionalitása minél több kihívás egy helyen elérhetővé tétele, melyekhez így a felhasználók könnyedén hozzáférnek, ezáltal nem kell az interneten hosszasan kutakodni egy-egy megfelelő kihívás megtalálásáért. A fejezet bemutatja a projekt funkcionalitásait szerepkörökre lebontva. Azért van szükség szerepkörökre, mert vannak az projektnek olyan funkcionalitásai, amelyekhez nem lenne jó, ha minden felhasználó hozzaférne. Ilyen például a kihívások módosítása vagy a felhasználók felfüggesztése, kezelése.

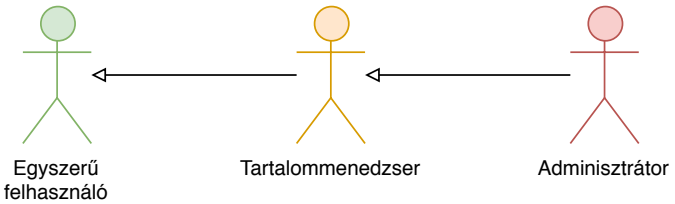
Egyes szerepkörök a 2. ábrán látható módon örökölik más szerepkörök jogosultságaiknak listáját és kiegészítik azt továbbiakkal. Három szerepkört különböztetünk meg:

- Az *Egyszerű felhasználó*knak van a legkevesebb joguk, csak a mobilalkalmazásba tudnak bejelentkezni, ahol minden mobil funkcionalitást elérnek.
- A *Tartalommenedzser*ek öröklik az egyszerű felhasználók jogait, használhatják a mobilalkalamzást, de ezen felül jogosultságuk van a webes felületen is bejelentkezni és hozzáférni az adminisztrációs funkcionalitások azon részeihez, amelyek a kihívások kezelését érintik.
- A legtöbb joggal az *Adminisztrátor*i szerepkör tagjai rendelkeznek, ők hozzáférnek minden elérhető funkcionalitáshoz.

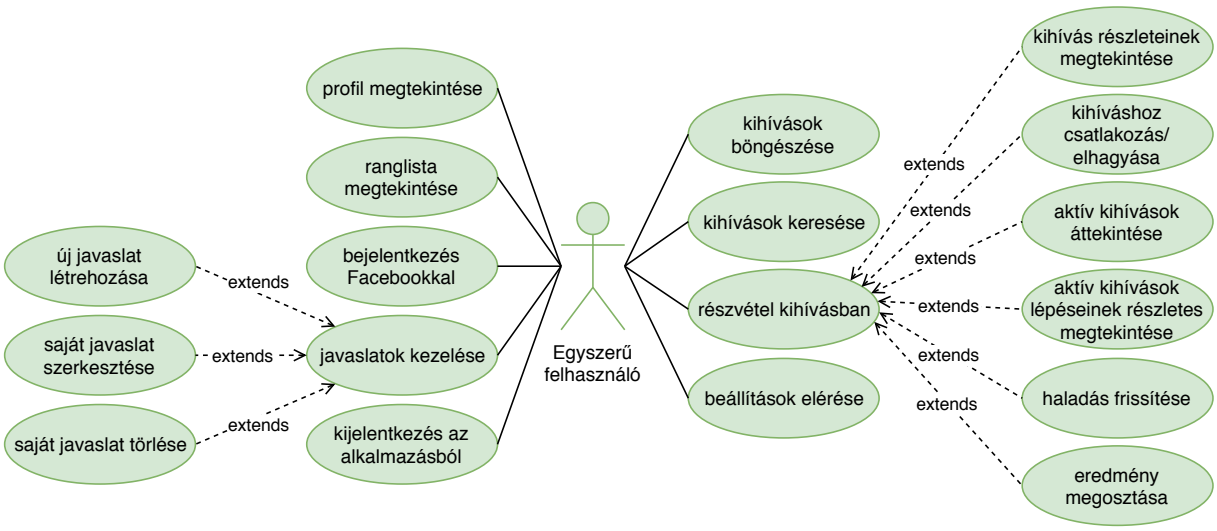
1.1. Egyszerű felhasználó jogosultságai

A mobilalkalmazás letöltése után minden felhasználónak, így az egyszerű felhasználónak (3. ábra) is szükséges bejelentkeznie az összes további funkcionalitás elérése végett.

Első nézetben át lehet tekinteni egy listában azokat a kihívásokat, amelyekben a felhasználó éppen részt vesz. Itt könnyedén lehet ellenőrizni, hogy melyik kihívás esetén milyen lépés vannak soron. Ebben a listában bármelyik kihívásra kattintva az aktuális lépés áttekintéséhez lehet eljutni, ahol további részleteket kaphat a felhasználó az aktuális (vagy akár az előző vagy következő) lépésről. A megnyitott ablakban előre és hátra lehet a lépések között járkálni, mindegyikről pontos leírás, utasításokat kapva. Ugyanitt lehet teljesítettnek jelölni a soron következő



2. ábra. Szerepkörök közötti öröklődés. A tartalommenedzser rendelkezik az egyszerű felhasználó jogaival is, az adminisztrátor pedig a tartalommenedzserével és az egyszerű felhasználóéval is.



3. ábra. Az egyszerű felhasználó számára elérhető funkcionalitások. Ezek kivétel nélkül mind a mobilalkalmazás által szolgáltatott funkciók.

feladatot, ezzel tovább haladva a kihívásban a cél fele. Ha a felhasználó befejez egy kihívást, akkor az alkalmazás gratulál neki, és kiír olyan információkat, mint a teljes teljesítési idő vagy a kihívással megszerzett összpontszám. Ezen kívűl lehetőséget biztosít az eredmények megosztására közösségi hálókon vagy más alkalmazásokon keresztül, ezzel is ösztönözve az ismerősöket, egy kihíváshoz való csatlakozásra.

Lehetőség van böngészni az elérhető kihívások között. Ezek egy listában jelennek meg, amelyben lehet keresni is a kihívás neve és leírása alapján, ezáltal mindenki megtalálhatja a számára legmegfelelőbbeket. Itt egy kihívást kiválasztva annak a részletei válnak láthatóvá, mint például az időtartama, leírása, a lépések összefoglalása. A felhasználónak lehetősége van, hogy a kiválasztott kihívásthoz csatlakozzon, vagy épp kilépjen belőle, valamint a kedvencei közé is adhatja, hogy majd később könyebben visszatérhessen rá.

Lehetősége van a felhasználónak a mobilalkalmazás bizonyos globális beállításait is megváltoztatni. Ezek közé tartozik a központi szerver elérési útvonalának módosítása, amely tesztelhetőségi előnyök mellett biztosíthatja a helyes működést esetleges szervermigrációk esetén is. Továbbá lehetséges a különböző telefonos értesítések engedélyezése vagy tíltása, így a felhasználó testreszabhatja hogy milyen történésekről szeretne értesülni.

Van mód új kihívás ötletek javaslására is a tartalommenedzserek felé. Lehet ajánlani egy alapötletet rövid leírással de lehetőség van ehhez konkrét lépéseket is küldeni. Ezzel, ha a felhasználó nem találná a számára megfelelő kihívást vagy csak egy jó elképzelése támad akkor jó eséllyel lehetősége lesz a koncepció kidolgozása után résztvenni a saját ötletére alapuló kihívásban. A beküldött javaslatok egy listában jelennek meg a feldolgozottsági állapotuk alapján. Bizonyos állapotokban az elküldött javaslatok még módosíthatók, kiegészíthetők, vagy ha a felhasználó meggondolja magát akkor törölhetők is.

Minden felhasználó megtekintheti a saját profilját, ahol egy listában megjelennek a befejezett kihívásai. Ezekre kattintva újra a gratulációs nézethez jut, ahonnan könnyedén végig nézheti újra az elvégzett lépéseket is. Egy másik listában szemügyre lehet venni a globális ranglistát és benne a felhasználó által elfoglalt helyet is, ezzel is motiválva a felhasználót, hogy minél több kihívás elvégzésével egyre több pontot szerezzen és magasabbra jusson a ranglistán.

Mindezek mellett, ha a felhasználó szeretne, ki is jelentkezhet az alkalmazásból, illetve ha úgy dönt, törölheti is a fiókját, törölve ezzel minden adatot róla és az elért teljesítményeiről.

1.2. Tartalommenedzser jogosultságai

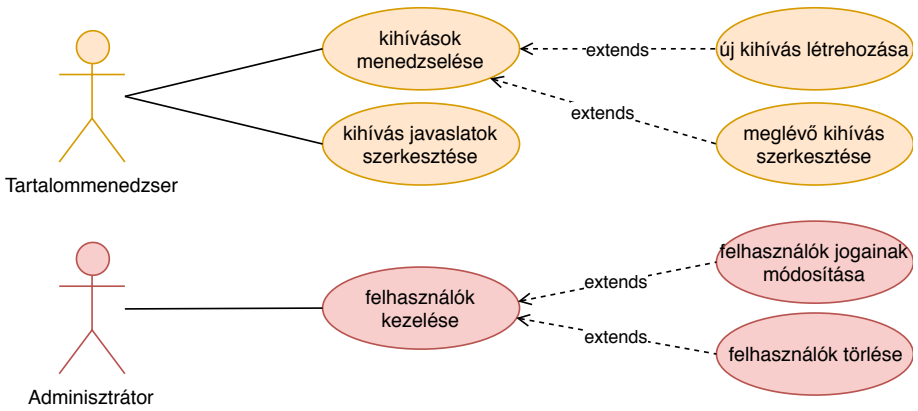
Egy tartalommenedzser jogú felhasználó (4. ábra) mindent megtehet, amit egy az 1.1 fejezetben bemutatott normál felhasználó és mindezek mellett jogosultsága van bejelentkezni a webes felület adminisztrátori részére is, ahol a kihívásokkal kapcsolatos műveleteket végezhet el azok bővítése és menedzselése céljából. A bejelentkezést itt is a Facebook fiók használatával lehet megtenni.

Ezen a felületen lehetősége nyílik kilistázni a közzétett kihívásokat, megtekintheti ezek részleteit és bizonyos részeit módosíthatja is. Ugyanitt van mód teljesen új kihívás létrehozására is.

Egy másik menüpont alatt a felhasználóktól érkezett különböző állapotú javaslatokat listázhatja. Ezeket szerkesztheti, kiegészítheti és ha már közzétételre kész állapotban vannak publikálhatja, ezzel gyarapítva az elérhető kihívások listáját.

1.3. Adminisztrátor jogosultságai

Az adminisztrátorok (4. ábra) az 1.2 fejezetben bemutatott tartalommenedzserek jogai mellett a felhasználókat érintő műveleteket is elvégezhetnek. Kilistázhatják őket, tartalommenedzseri és adminisztrátori jogokat adhatnak, ezzel gyarapítva a háttérben tevékenykedő emberek számát, de akár el is vehetik ezeket a jogokat.



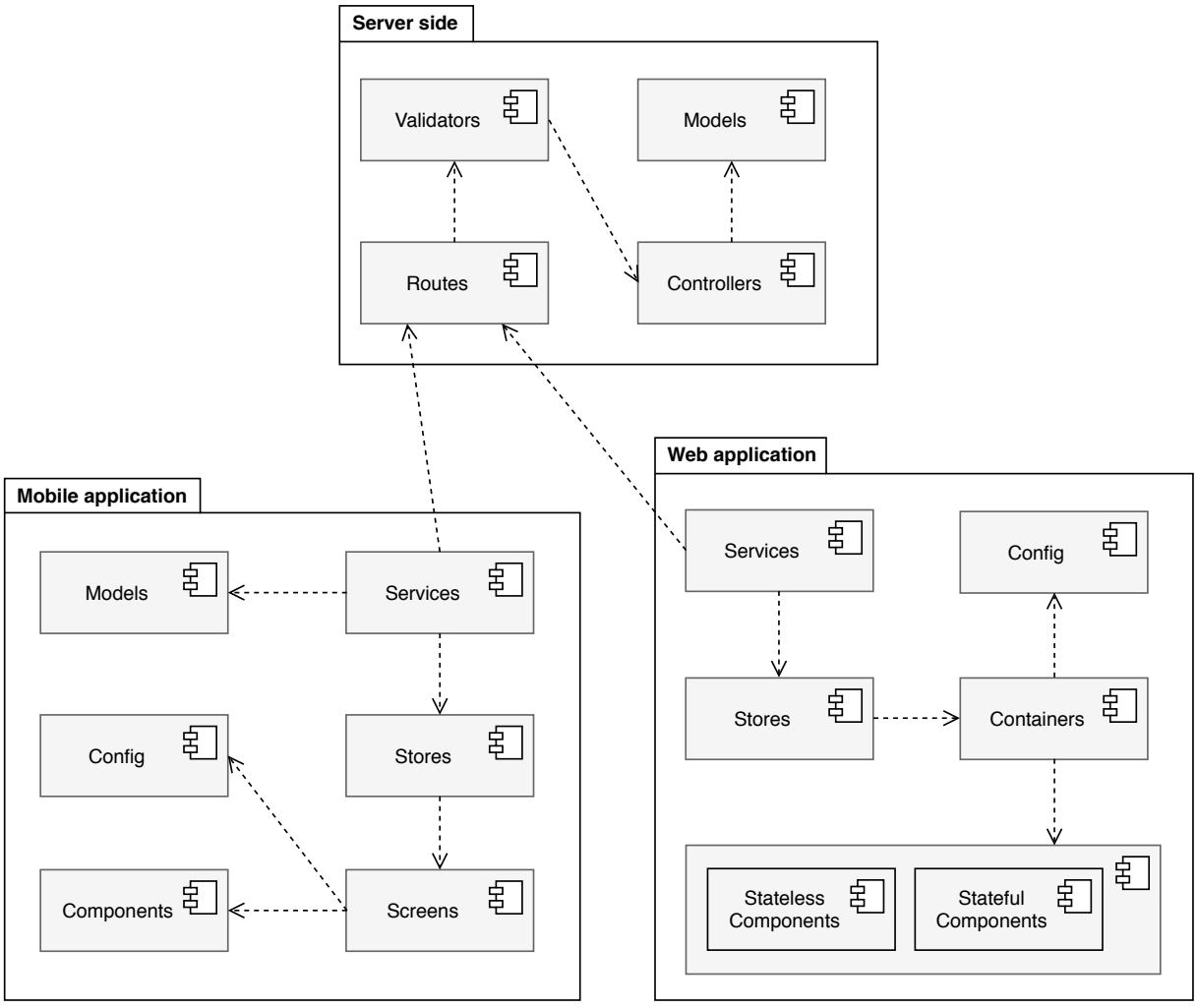
4. ábra. A webes felület által nyújtott funkcionalitások melyek a tartalommenedzserek és az adminisztrátorok számára érhetőek el.

2. Architektúra

A rendszer három fő komponensből épül fel a 5. ábrán látható módon. A mobilalkalmazás és a webes felület, melyeket a harmadik egység, a központi szerver szolgál ki. Mindegyik esetében megfigyelhető a többrétegű architektúra, ezzel létrehozva egy könnyen fejleszthető és karbantartható rendszeregyüttest, melynek javításához vagy a bővítéséhez elég bizonyos rétegeket módosítani az egész komponens módosítása helyett. Az ilyen réteges szerkezetek esetén minden réteg csupán a szomszédjaival folytat kommunikációt.

A szerver esetében a *routes* réteghez érkeznek a kérések, melyeket ez továbbít a validáló (*validators*) réteg számára. Itt a kérésekkel kapcsolatos bizonyos érvényesítések hajtódnak végre, és ha a kérés megfelelő és teljesíthető, akkor átveszi a megválaszolását a *controllers* komponens. Itt, felhasználva a modell (*models*) rétegben definiált adatsémákat, végrehajtódnak a megfelelő adatmanipulációs műveletek és egy válasz kerül visszaküldésre a kérést intéző kliens számára.

A két kliens felépítése hasonló. Mindkettő a *services* rétegben definiált kérések segítségével



5. ábra. A rendszer komponensei és azok kapcsolatai egymással.

indít kommunikációt a központi szerver fele. Ehhez a mobilalkalmazás felhasználja a modell (*models*) rétegében meghatározott sémákat, amelyekkel ellenőrizni tudja a válaszként kapott adatok felépítését. A services rétegből az adatokat átveszik a *stores* komponensben létrehozott osztályok, melyek már a megjelenítési réteghez tartoznak, egyszerre töltik be a modell és a vezérlő szerepet az *MVC* [23] modellből. Ettől a ponttól kissé eltér a két kliens felépítése, a mobilalkalmazás esetében a (*screens*) rétegben található implementációk felelősek a telefon képernyőjén megjelenő nézetek létrehozásáért. Ehhez felhasználják a komponens (*components*) rétegben található különböző általunk létrehozott grafikai elemeket és a konfiguráció (*config*) rétegben fellelhető, a nézetek közötti navigációs lehetőségeket definiáló fájlokat. A webes alkalmazásnál nem screens, hanem konténer (*containers*) réteg látható, amelyben az ugyanolyan környezetben előforduló oldalaknak (például az adminisztrációs felületen fellelhető oldalak) vannak definiálva konténerek, melyek a minden oldalon megjelenő azonos elemeket (navigációs sáv, fejléc, lábléc) és az oldalak közötti navigációt biztosítják. Ezeken a konténereken belül különböző állapottal rendelkező (*stateful*) és állapot nélküli (*stateless*) komponensek váltogatják egymást, melyek az oldal további látható részeit alkotják.

A 6. ábrán látható, hogy a bemutatott különböző komponensekből felépülő rendszerek milyen módon kommunikálnak egymással. Minden kliens minden fajta kérését a *Web szerver*nek küldi el, ahonnan a *REST* kérések továbbítva vannak az *API szerver* fele. A statikus adatokra vonatkozó kéréseket (mint például a webes alkalmazás kérése) a Web szerver által kerül kiszolgálásra.

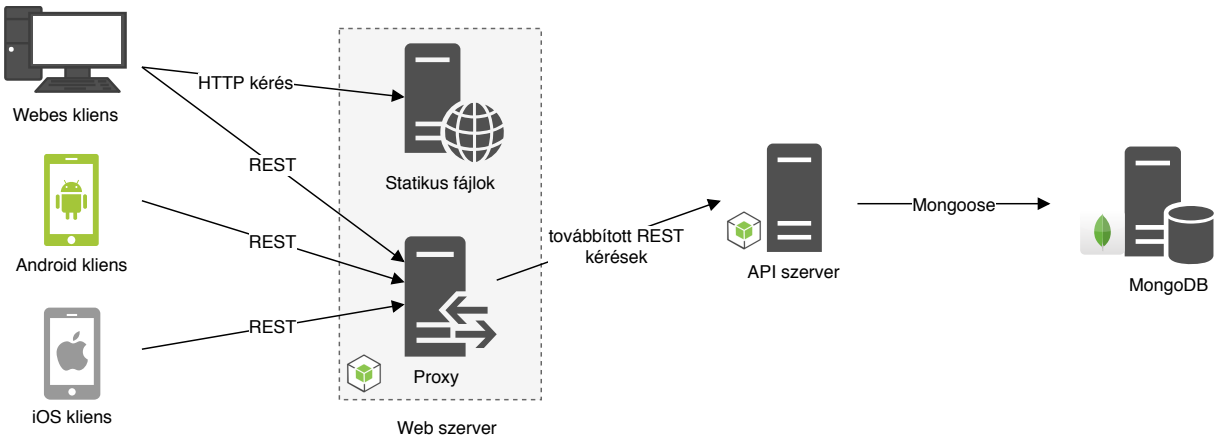
Ennek megvalósítására egy *Proxy*-t használunk, amely a beérkező kérések URL-jeinek mintája alapján el tudja dönteni, hogy szükséges a továbbirányíáts vagy sem. Több okunk is volt ilyen módon megoldani a kérések kezelését. Első sorban így csak egy portot kell megnyitsunk a külvilág fele, ezzel csökkentve a biztosnági kockázatokat. Emellett ezzel orvosoltuk a *Cross-Origin* hívások [30] problémáját is, ami akkor lépne fel, ha a böngészőben egy adott URL-ről betöltött oldal egy másik URL-t akarna használni valamilyen kommunikáció megvalósításához. Nem utolsó sorban így van egy moduláris API-nk, amit lehetne akár más nyelvben megírt változattal is helyettesíteni.

Az API szerver a beérkezett REST kérése kiszolgálása érdekében kommunikációt folytat a rendszerünket kiszolgáló adatbázissal.

2.1. Szerver

2.1.1. Adatmodell

A központi szerver által kezelt adatok egy dokumentum alapú *NoSql* [31] adatbázisban vannak tárolva, ami a 3.1 fejezetben kerül kifejtésre. Az adatbázisban úgynevezett kollekciókban,

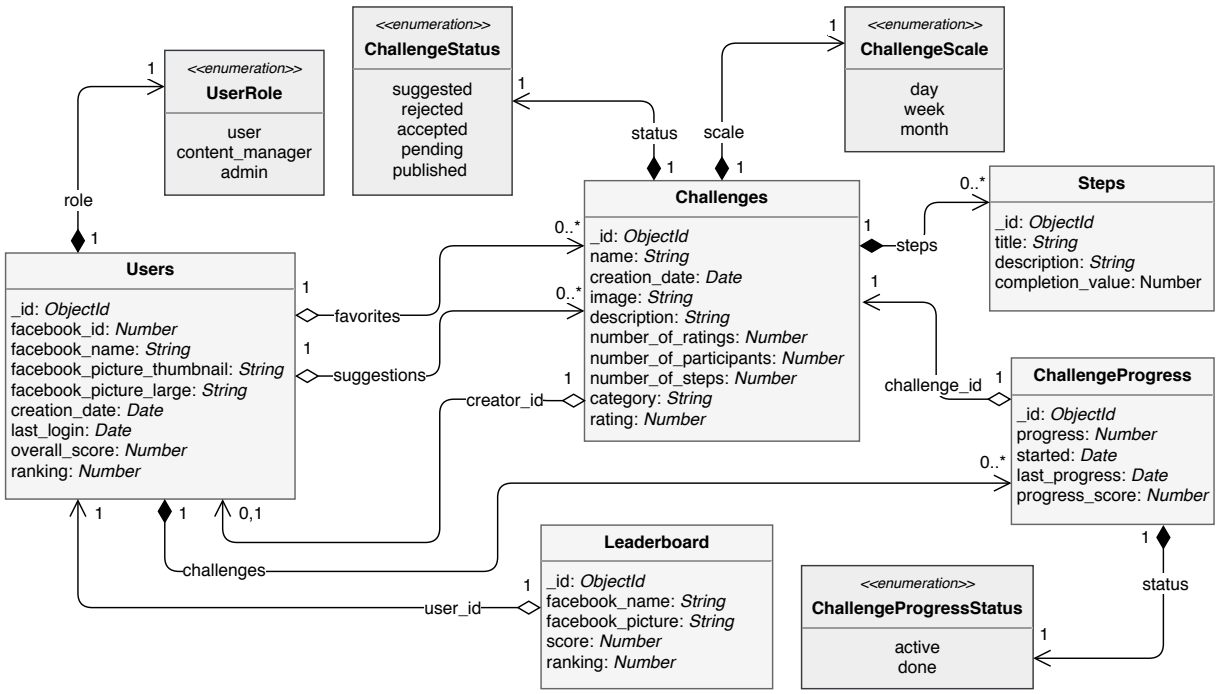


6. ábra. Kommunikációs diagram, mely bemutatja, hogy a különböző kliensek és egységek között hogyan történik az információcsere.

azokon belül dokumentumokban vannak tárolva az adatok, ezeket a relációs adatbázisok tábláihoz és soraihoz lehetne hasonlítani. A szerveren a *models* komponenseben vannak definiálva azok a sémák, amiket az entitásaink kell kövessenek. Itt fel vannak sorolva a különböző kollekciókban használt mezők, azoknak a típusai, bizonyos esetekben alapértelmezett értékei vagy lehetséges értékei, valamint egy mező kötelezően kell-e szerepeljen egy dokumentumban.

Ahogy az a 7. ábrán is látható három különálló kollekciónk van, *Challenges*, *Users* és *Leaderboard*, valamint további két séma van definiálva, a *Steps* és *ChallengeProgress*, amelyek be vannak ágyazva más kollekciók mezői közé. A kihívásoknak lehetnek lépései így a Challenges dokumentumoknak van egy Steps sémákat tartalmazó tömbje, valamint a felhasználók csatlakozhatnak kihívásokhoz, ilyenkor hasonlóan egy beágyazott ChallengeProgress sémában kerül követésre az adott kihívásban való előrehaladás állapota.

A Users kollekcióban kerülnek tárolásra a rendszerbe Facebook fiókjukkal bejelentkezett felhasználók. Eltárolásra kerülnek a Facebooktól kapott adatok, mint az ID, név és kép. A maradék mezők alapértelmezett értéket kapnak, minden felhasználó indulásból *user* azaz a 1.1 fejezetben bemutatott egyszerű felhasználói jogosultságot kap. A *favorites* és a *suggestions* mezők üres tömbök lesznek, ezek fogják a későbbiekben tárolni azoknak a kihívásoknak az azonosítóit, amelyeket a felhasználó a kedvencei közé tesz vagy javasol. Az *overall_score* mezőbe van számon tartva a felhasználó pontszáma, ami nulláról indul, valamint a *ranking* adattag felelős a felhasználó a ranglistában elfoglalt helyének tárolásáért. A *challenges* mező is egy üres tömb lesz a kezdetekben, de itt nem azonosítók lesznek a továbbiakban tárolva, hanem beágyazott *ChallengeProgress* sémák, amelyekben számon lesz tartva azoknak a kihívásoknak az állapota, amelyekbe a felhasználó részt vesz vagy már befejezett. Itt tárolva van a kérdéses kihívás azonosítója, az hogy épp teljesítés alatt van, vagy már be lett fejezve, a csatlakozás időpontja, az eddig a kihívással összegyűjtött pontszám, a soron következő lépés száma valamint a legutóbbi lépés teljesítésének ideje.



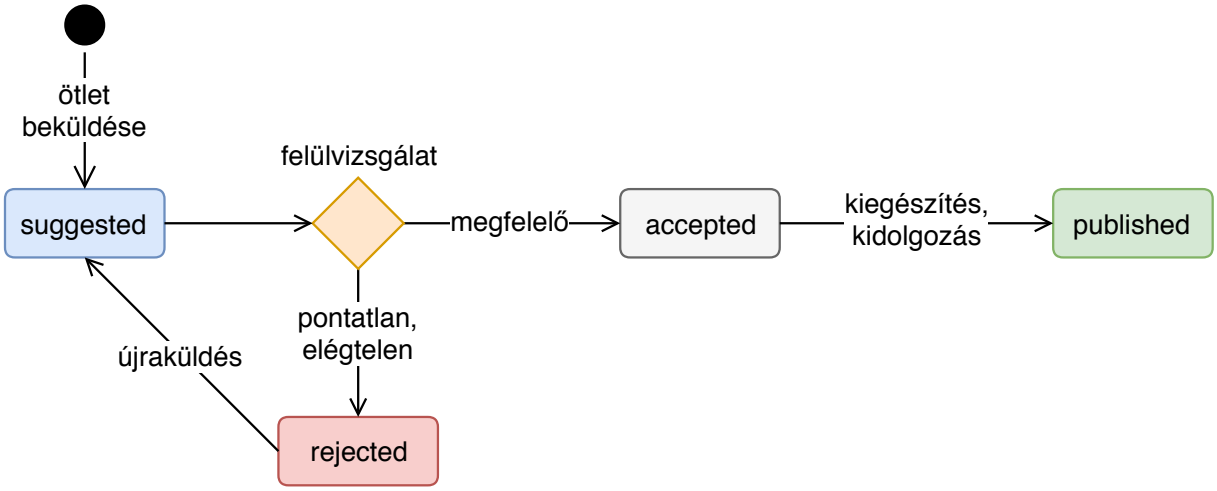
7. ábra. Az adatbázisban lévő kollekciók és beágyazott kollekciók sémái, valamint a felhasznált felsorolás típusok.

A Challenges kollekció tartalmazza a mindenki számára elérhető és a különböző állapotú beajánlott kihívásokat is. Megkülönböztetésükre egy *status* nevű mezőt használunk, ami több értéket is felvehet: *suggested*, *rejected*, *accepted*, *published*, a 8. ábrán látható módon. Ezek közül az utolsó, publikált státusszal rendelkező kihívások érhetőek el minden felhasználó számára. Ezen kívül több minden is tárolva van a kihívásokról, azok neve, leírása, létrehozási ideje, képe, az értékelők-, résztvevők- és a lépések száma, a kategóriája, összesített értékelése, ajánlójának (ha van) az azonosítója, lépései és az egyenként a lépéseknek a teljesítésére szánt időmennyiség mércéje (*scale*), ami lehet nap, hét és hónap. A lépéseknél itt is beágyazott sémáról beszélhetünk, egy tömbben vannak a Steps séma szerint meghatározott adatok tárolva. Minden lépés rendelkezik egy címmel, leírással és egy a teljesítéséért járó pontmennyiséggel.

Az utolsó Leaderboard kollekció tárolja annak a 100 felhasználóknak a fontosabb adatait, akik a legmagasabb pontszámmal rendelkeznek. Ennek a kollekciónak a tartalma bizonyos időközönként automatikusan frissül, amikor a rendszer újra rangsorolja a teljesítményeket, figyelembe véve minden felhasználó összpontszámát.

2.1.2. Adathozzáférési réteg

Az API szerver a MongoDB adatbázissal való kommunikációt a 3.1 fejezetben leírt Mongoose segítségével valósitottuk meg. A Mongoose lehetőséget adott számunkra hogy a szerveren objektumokkal végezhessünk műveleteket, amelyek megfelelnek az adatbázisunk dokumentumainak. Mongoose modelleket hozzunk létre, amelyek segítettek a kihívások és a felhasználók



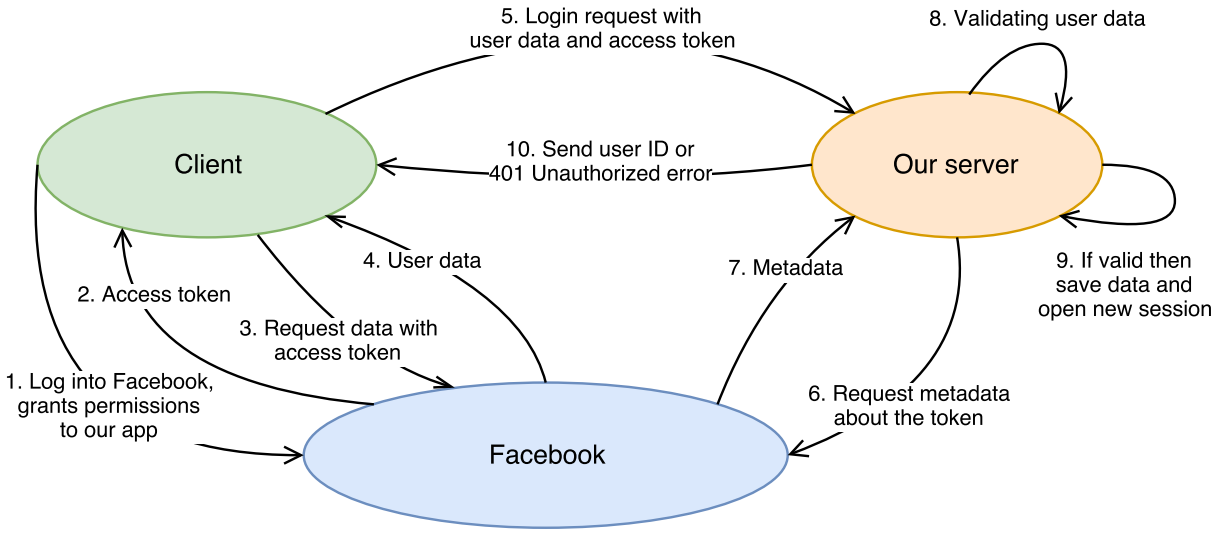
8. ábra. Egy kihívás több különböző állapotban is lehet. Az ajánlás pillanatában még csak *suggested*, de ha elfogadásra kerül és a tartalommenedzserek teljes értékű kihívást készítenek belőle akkor *published* állapotba kerülhet, ezzel minden felhasználó számára elérhetővé válva.

adatainak megfelelő kezelésében és ellenőrzésében. A modellekben az adattagok rendelkeznek típusokkal, alapértelmezett értékekkel és olyan dekorátorokkal mint például a *required*, amellyek kötelezővé tudtunk tenni egy adattagot. Az adatbázisho való hozzáférés egyszerű parancsokon keresztül történik, nincs szükség MongoDB scriptek írására. Az adatmanipulációt CRUD műveleteken keresztűl végesztük. A *Create, Read, Update, Delete*), röviden CRUD, a négy alapművelet, melyel egy perzistens api-nak dolgoznia kell. A *request-response* komunikáció méretének kordábab tartása és a memóriahasználat kontroll nélküli növekedésének elkerülésére a Mongoose *paginate* szolgáltatását használtuk.

2.1.3. Biztonság

Ahogyan az 1.1 fejezetben is említve volt, az alkalmazás használatba vételéhez (legyen az a mobil vagy a webes kliens) először be kell jelentkezzen a felhasználó a Facebook fiókjával. Ez a bejelentkezési folyamat a háttérben több lépésből áll, ahogy azt a 9. ábra is szemlélteti.

Első lépésként az alkalmazás átirányít a bönégszőbe vagy a telepített Facebook alkalmazáshoz és a felhasználó bejelentkezését fogja kérni. Ezek után az alkalmazás átal igényelt jogosultságokat kell a felhasználó elfogadja. Mindezek után egy kérés indul a Facebook szervere fele, ahonnan válaszul egy meghatározott időre érvényes hozzáférési kulcs fog érkezni. Következő mozzanatként a megszerzett kulcsot felhasználva a kliens alkalmazás lekéri a felhasználóról az igényelt adatok (Facebook ID, név, kép). Mindezek után a kliens küld egy kérést a saját szerverünk fele, amiben elküldi a megszerzett adatokat és a hozzáférési kulcsot. A kérést megkapva a szerver le fogja ellenőrizni a Facebook ID-t és a hozzáférési kulcsot. Ehhez a Facebook biztosít egy API endpoint-ot, ahol hitelesítve a kérést az alkalamásunk adataival információkat lehet lekérni egy hozzáfárási kulcsról. Ellenőrizhető, hogy a kulcsot a mi alkalmazásunk bocsájtotta-



9. ábra. Bejelentkezés folyamán a kommunikáció a kliens, a facebook szerver és az alkalmazás szervere között.

e ki, hogy a klienstől kapott Facebook ID van-e a kulcshoz társítva illetve, hogy egyáltalán érvényes-e még a kulcs. Ha minden helyes, akkor a felhasználónak hozzáférése nyílik az alkalmazásunk használatához. A szerver attól függően, hogy új vagy már létező felhasználóról van szó elmenti vagy frissíti az adatokat és egy új *session*t indít, amiben eltárolja a felhasználónak a szerverünktől kapott azonosítóját, a szerepkörét valamint a kliens *user agent* információit. Utolsó mozzanatként visszaküldi a szerverünktől kapott azonosítót, aminek segítségével a felhasználó el tudja majd érni a saját adatait.

Hitelesítés utána a kliensek szabadon használhatják a szerverünk által biztosított endpointokat. A bejelentkezési kérésen kívűl minden másnál megtörténik a session információk ellenőrzése. Kell hogy legyen session, és tartalmazni kell az előző bekezdésben leírt értékeket, valamint a kérés fejlécében szereplő user-agent információk is meg kell eggyezenek a session-ben tárolt, bejelentkezésnél kapott adatokkal.

Ha a session információk rendben vannak akkor a szerver ellenőrzi, hogy a felhasználónak a szerepköre alapján van-e joga elvégezni a kért műveleteket. Erről egy saját szerepkör alapú hozzáférést szabályozó rendszer (*RBAC*) gondoskodik. Minden szerepkör esetén meg van határozva, hogy a hozzá tartozó felhasználók milyen feltételeket mellett milyen műveleteket végezhetnek el. Emellett egy szerepkör örökölhet jogokat más szerepköröktől is. Ezek alapján példányosításkor az RBAC felépít egy *roleMap*-et, amit majd annak eldöntésére használ, hogy egy bizonyos szerepkörből való felhasználó a megadott paraméterek mellett elvégezheti-e a kért műveletet. Ehhez a rendszer biztosít egy *can* függvényt, ami a szerepkör jogai és az örökölt jogok alapján eldönti, hogy egy művelet egy felhasználó számára elvégezhető-e vagy sem. Ha elvégezhető akkor folyik tovább a végrehajtás, ha nem akkor egy *403 Forbidden* státuszú üzenet kerül visszaküldésre a felhasználónak.

2.1.4. API

A Daily Challenge projetkt keretében elkészült API szerver egy RESTful, publikus szolgáltatásként üzemel. Ez szolgálja ki a mobil alkalmazás és a webapplikáció által küldött HTTP kéréseket. A kommunikáció HTTP protokollon alapszik, JSON formátumú üzenetekkel.

Az API szerver legfontosabb elérési pontja a kihívásokat kezelő */challenges*. Ezen keresztül legetőség nyílik a kihívások lekérésére és szűrésére, valamint kihívás ötletek beküldésére. Ha csak egyetlen kihívás adatait szeretnénk kezelni, akkor a */challenges/:challengeID* formátumot használhatjuk. A */challenges/:challengeID/steps* hozzáférést használva az egyes kihívások különböző lépéseihez férhetünk hozzá. Hasonlóan mint a kihívásoknál, a */:stepID* hozzáfűzésével pontosíthatjuk az elértni kívánt lépést.

A másik lényeges elérési pontja az API szervernek a felhasználók kezelésére létrehozott */users*. Itt fogadja a szerver a bejelentkezési kísérleteket és ide futnak be a kijelentkezési kérések a */users/logout* címre. A megfelelő felhasználó azonosítójának hozzáfűzésével, elérhetjük a részletes adatait a */users/:userID* elérés használva. Erre csak akkor van lehetőségünk, ha a saját adatainkhoz szeretnénk hozzáférni, vagy ha adminisztrátor jogokkal rendelkezünk. A felhasználók által felvett kihívásokat a */user/:userID/challenges/:challengeID* elérést használva tudjuk kezelni.

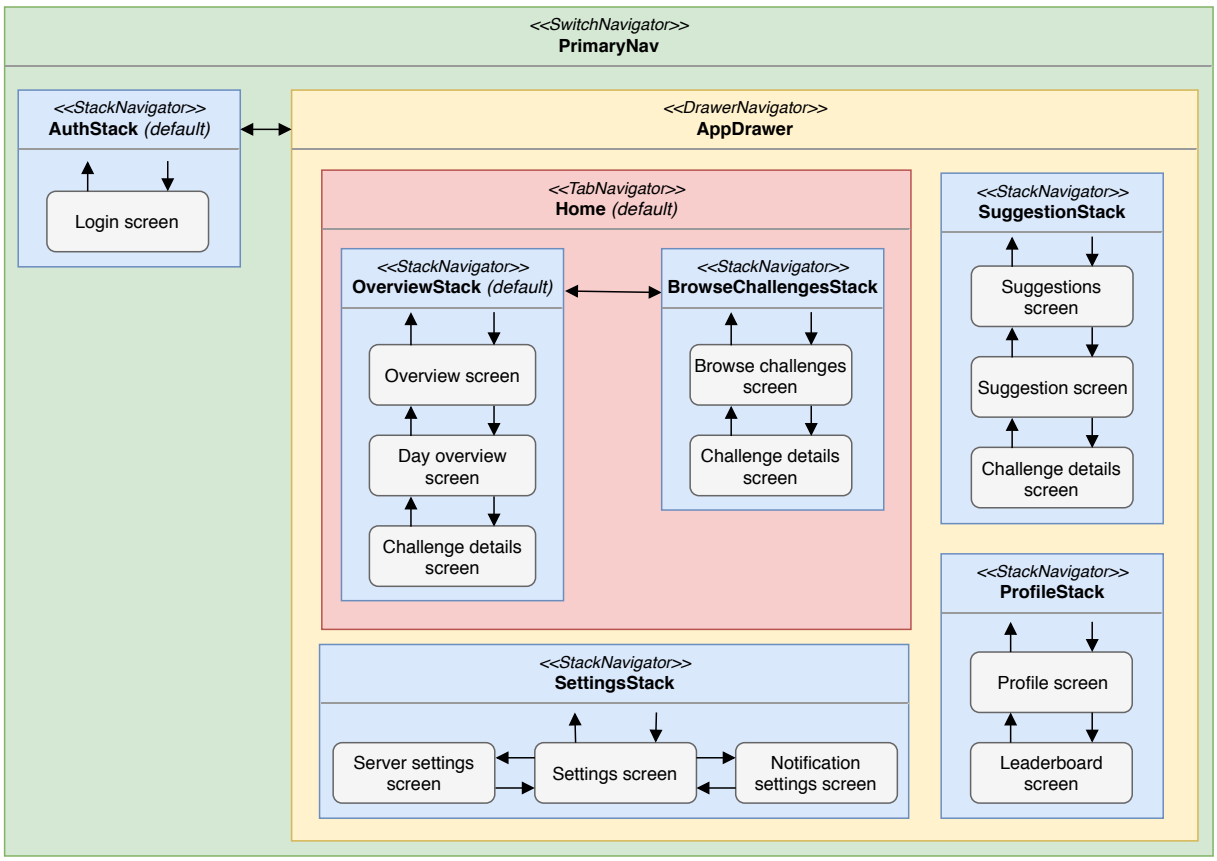
2.2. Mobil

2.2.1. A felhasználói felület

A React egy komponens alapú könyvtár, segítségével komponens alapú nézeteket lehet létrehoni. Egy komponens egy elemi darabkája, építőeleme a felhasználói felületnek, mint például egy gomb vagy egy beviteli mező. Ezek a komponensek testre szabhatók, újra hasznosíthatók és egymásba ágyazhatók, egy komponens tartalmazhat más komponenseket vagy összetettebb elemeket. A React gondoskodik a felület fürge működéséről is, felismeri, hogy mely komponenseket kell újra rajzolni az adatváltozásoknak megfelelően, és csak azokat a részeket frissíti be.

Az alkalmazásban mindent React komponensek [16] és ezeket kiterjesztő osztályok alkotnak. A felhasználó által látott nézetek is ilyen osztályok, melyek sok kis komponensből épülnek fel. Ezek az osztályok a projekten belül a screens és components rétegben vannak definiálva (5. ábra).

A felhasználói élményt növelendő és a saját munkánkat megkönnyítendő a Native Base [10] nevű felhasználói felület eszköztár használata mellett döntöttünk. Ezzel az alapvető React Native komponensek [11] mellett (View, Text, stb.) egy új eszköztárat kaptunk sok új hasznos komponenssel [9]. Így a végleges alkalmazás felhasználói felületének legnagyobb részét Native



10. ábra. Az alkalmazáson belül a nézetek közötti navigálási lehetőségek felhasználva a négy bemutatott navigációt kezelő komponenst.

Base elemek alkotják.

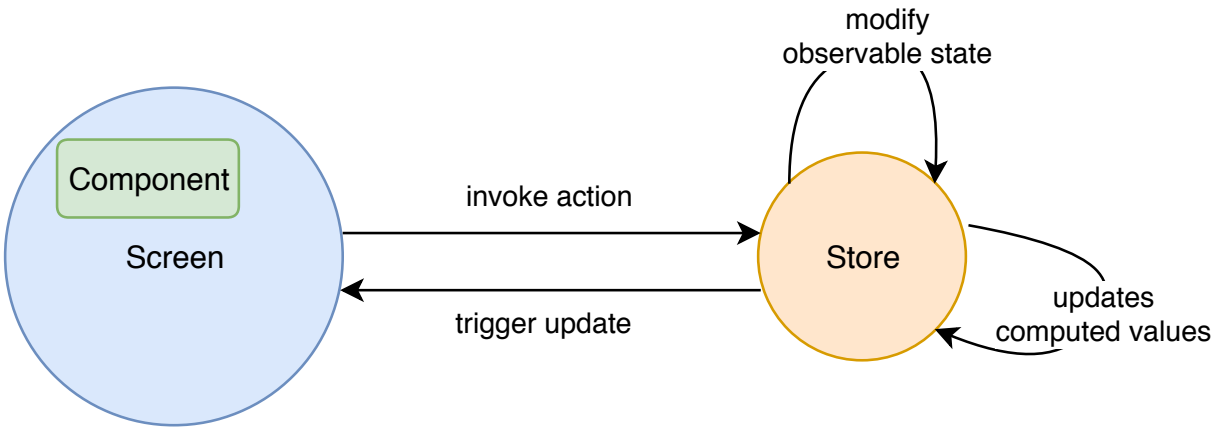
A komponensek testre szabására a CSS-hez hasonló stíluslapok (StyleSheet) [13] segítségével történik. A legtöbb CSS-ben létező stílusnév és érték használható, annyi különbséggel, hogy a React-ben konvenció szerint CamelCase-el kell írni a stílusneveket, például *marginTop* *margin-top* helyett. Minden komponensnek van egy *style* nevű property-je, itt megadva különféle stílusokat lehet megváltoztatni ezek kinézetét, pozícióját és méretét.

A nézeteket megvalósító osztályok között a 10. ábrán látható módon van lehetősége navigálni a felhasználónak. Az alkalmazáson belüli navigációról egy *React Navigation* [15] nevű könyvtár gondoskodik. Ez több féle lehetőséget is biztosít a nézetek közötti lépkedés megvalósítására, ezekből négy van használatban a mobilalkalmazásunkban is: *StackNavigator*, *TabNavigator*, *DrawerNavigtor*, *SwitchNavigator* [14].

2.2.2. Megjelenítési réteg

Az alkalmazás megjelenítési rétege három komponensből áll: *screens*, *components* és *stores* (5. ábra) . Ezekből az első kettő felelős a nézetért (view) és a harmadik pedig egyben a modell és a vezérlő (model, controller).

A *screens* tartalmazza azokat a nézeteket, amelyeket az alkalmazásban való navigálás so-



11. ábra. Adatáramlás a megjelenítési rétegben

rán lehet látni. Ezek a nézetek a modellek által nyújtott adatokat jelenítik meg megfelelően formázva, valamint képesek a felhasználóval való interakcióra is gombokon vagy más elemeken keresztül. A *components* tartalmazza azokat az összetettebb saját elemeket, amelyek több nézetben is előfordulnak, így elégséges egyszer definiálni őket, majd innen használni minden alkalommal, amikor szükség van rájuk.

Az alkalmazás kinézetének a modelltől és vezérléstől való elválasztásáról a 3.2 fejezetben bemutatott MobX állapot menedzsment rendszer gondoskodik. Ezáltal minden nézethez egy *Store*-nak nevezett elem társul, ami a képernyőn megjelenített adatokkal foglalkozik és a változásokra reagálva, manipulálja az adatokat vagy kommunikál a központi szerverrel. A *screens*-ben található nézetek nem foglalkoznak adatmanipulációval, csak a hozzájuk rendelt állapot menedzselő osztályok által nyújtott adatokat jelenítik meg valamint interakció esetén a megfelelő függvényeket hívják meg.

A store-okban @*observable* annotációval vannak ellátva azok az adattagok, amelyekről a nézetek kell tudjanak, meg kell jelenítsenek. Maguk a nézetek @*observer* azaz megfigyelő annotációt viselik, így értesülnek a megfigyelhető adattagok változásairól és ilyenkor újra rajzolják a megfelelő részeket. Ahhoz, hogy egy store értesíteni tudja a megfigyelőket a változásokról azokat a függvényeket amelyekben megfigyelhető adattagok változhatnak @*action* annotációval kell ellátni. Ez biztosítja azt, hogy a változtatásról biztos tudomást szerezzenek a nézetek. Ezeken kívül a @*computed* annotáció használható olyan *JavaScript getter*-eknél [25] amelyek értéke fel van használva a nézeteken és ez a szolgáltatott érték több adattagot felhasználva határozódik meg. Ilyen esetben a @*computed* annotáció biztosítja, hogy bármely felhasznált adattag változásának hatására újraszámolódjon a szolgáltatott érték és a nézet értesítve legyen erről.

Interakció esetén a store-oban definiált függvényeket hívják meg a nézetek, ahol végrehajtódnak a megfelelő műveletek, az adatok manipulációi, valamint az esetleges computed értékek újra számolódnak, majd erről egyből értesülnek is a nézetek és újra rajzolják a megfelelő részeket. Ezt a körforgást szemlélteti a 11. ábra is.

2.2.3. Üzleti logika

Az üzleti logika a mobilalkalmazásban két réteget foglal magába, a modell (*models*), ahol a szervertől érkező adatok modelljei vannak meghatározva, valamint a *services*, amelyben a szerver fele irányuló kérések vannak definiálva. Az megjelenítési réteg külön van választva az üzleti logikától, a store-okból nem indul direkt kérés a szerver fele. Minden store-ban van egy példány a szükséges osztályokból az üzleti logika rétegről, és ezeknek a függvényeit használva történik meg a kommunikáció. A kérésekre a válaszok még itt a services rétegben ellenőrizve vannak. Ha a szervertől érkező válasszal minden rendben, akkor megtörténik a modell rétegben található sémák alapján az adatintegritás ellenőrzése. Ha az adatok felépítése is rendben van, akkor visszatérülnek a body-ban lévő információk JSON [26] formában. Ha valamilyen hiba lép fel és a kérésre érkező válasz nincs rendben, akkor a felmerülő hibának megfelelően az Error [24] objektum egy példányát dobja vissza a függvény, amit a felsőbb rétegek kapnak el és kezelnek le megfelelően. Több saját az Error-ból származó osztályt is definiáltunk, mint például *NotFoundError* ha egy kért erőforrás nem található és 404-es hibakódot ad vissza a szerver, vagy *ObjectSchemaError*, ha az érkezett adatok nem felelnek meg a modell rétegben definiált felépítésnek.

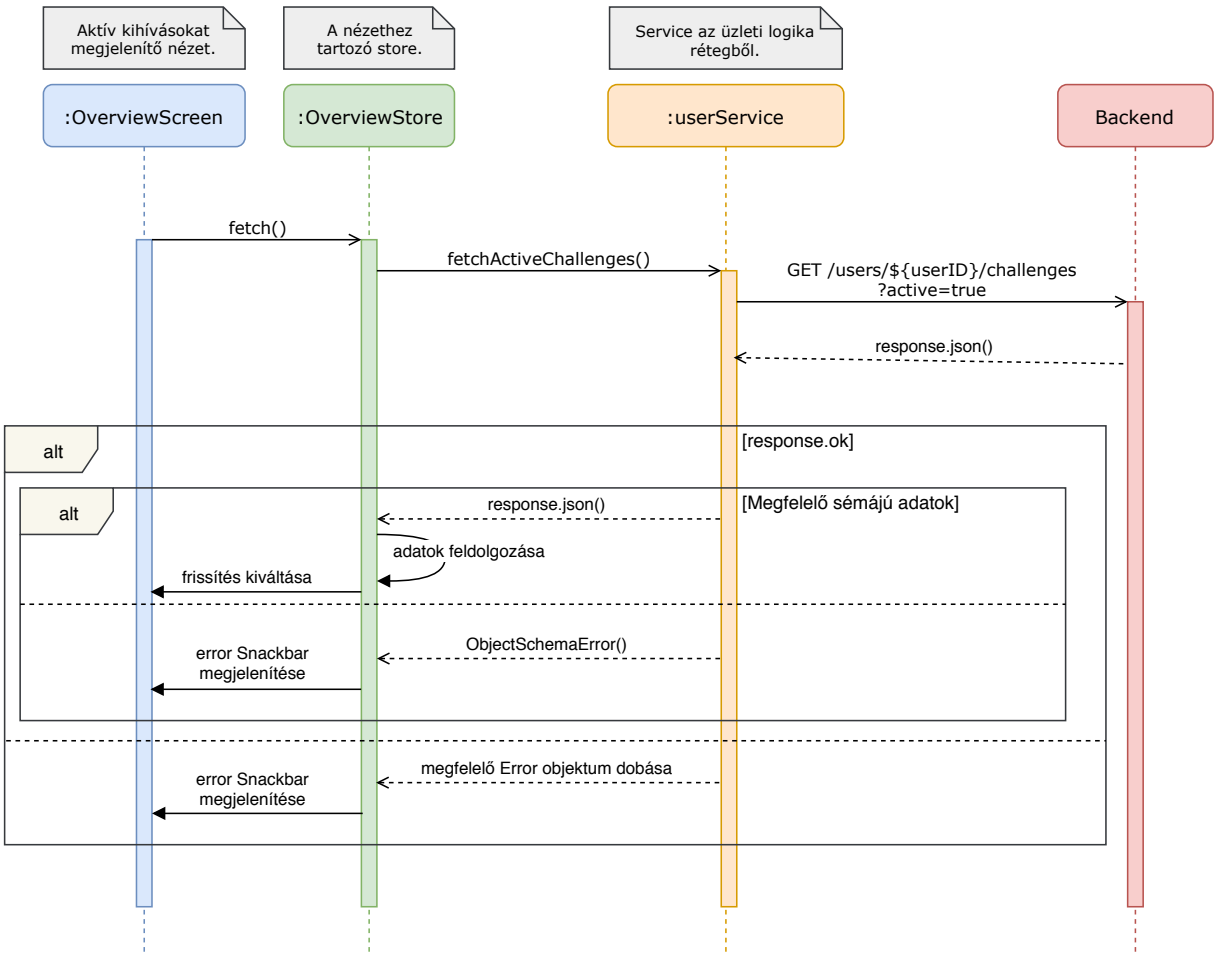
2.2.4. Kommunikáció a szerverrel

Az alkalmazás REST kéréseken keresztül kommunikál a központi szerverrel. A kéréseknek az implementációi, ahogy már a 2.2.3 fejezetben említve volt, a service könyvtárban találhatóak különböző fájlokban (userService, challengeService, stb.).

A szerver eléréséhez a Mozilla [8] által nyújtott Fetch API-t [28] használtuk. Ezzel lehetőségünk van aszinkron hálózati kéréseket küldeni és válaszokat fogadni. Ezt a Fetch API elterjedése előtt a XMLHttpRequest (XHR) [32] használatával lehetett elérni. A Fetch API-val ellentétben az XHR nem tartja be a Separation of concerns (SoC) elvet, a bemenet, kimenet, az állapot mind egy objektumban van menedzselve valamint az állapotok eseményekkel vannak kezelve, ami nem egyezik a JavaScript jelenlegi irányelveivel, amely a Promise [27] használatát helyettesíti előnybe Callback [22] helyett. Így Fetch egy tisztább és átláthatóbb API-t biztosít az XHR-el szemben.

A Fetch-nek egy kötelező paramétere van, az elérési út (URL) a kívánt erőforráshoz. Opcionálisan megadható egy második paraméter is, ami segítségével testreszabható a HTTP kérés. Itt határozható meg a kérés típusa, ami alapértelmezetten GET, valamint megadhatunk header és body paraméterek is.

A hálózaton keresztüli kommunikációk aszinkron működnek, a Fetch kérés egy Promise-t térít vissza, ami egyszerűvé teszi ezeknek az aszinkron válaszoknak a kezelését és feldolgozását.



12. ábra. Kérések általános kezelése kliens oldalon. Több rétegen való áthaladás után a kérés elér a szerverhez, ahonnan az információk visszaküldése után a kliens ellenőrzi ezeket és ennek fényében felhasználja az adatokat vagy hibaüzenetet jelenít meg.

A 12. ábrán egy szekvencia diagram látható a, amely bemutatja a 2.2.2 és 2.2.3 fejezetekben ismertetett megjelenítési és üzleti logika rétegek kommunikációját a központi szerverrel.

2.3. Web

A webalkalmazás az első megnyitáskor a böngésző betölti a Webpack (3.2) által felépített JavaScript csomagot, amely az oldal megjelenítését teszi lehetővé. A webes rész architektúrája nagyban megegyezik a 2.2 fejezetben bemutatott Mobilalkalmazás architektúrával. A továbbiakban csak a lényegesebb különbségeket emeljük ki.

2.3.1. Single Page Application

A *Single page alkalmazások* (Egy oldallal rendelkező alkalmazások) lényege, hogy a felhasználóval történő kommunikáció során nem töltődnek be új oldalak, hanem az egyetlen már előre betöltött oldal elemei változnak dinamikusan. Ez a megközelítés megszakításoktól mentessé teszi a felhasználói élményt. Az összes szükséges kód, legyen az JavaScript, HTML vagy

CSS, vagy az oldal első megjelenésekor töltődik be a felhasználó böngészőjébe, vagy a megfelelő adatok dinamikusan töltődnek, amikor szükség van rájuk. A 3.2 fejezetben bemutatott React.js nyújtott egy tökéletes alapot a webalkalmazásunk ilyen módon való felépítéséhez.

2.3.2. Responsive Design

A 3.2 fejezetben bemutatott Bootstrap lehetőségeit kihasználva optimalizáltuk az webalkalmazásunk megjelenítését minden képernyőtípusra. Nagy kijelzőn való megtekintés esetén a kezelőpult bal oldalán egy folyamatosan látható oldalsó menü található. Emellett egy teljes hosszúságban elhelyezkező fejléc gondoskodik arról, hogy a felhasználó mindíg láthassa, melyik fő oldalon tartózkodik. Csökkentett képernyőméret esetén a kihívások áttekintése megváltozik, a méret függvényében kevesebb illetve több oszlopban történik a kilistázás. Ha a méretet lecsökkentjük a mobil kinézet határa alá, akkor az oldalsó menü egy összecsukható menüvé alakul a bal oldalon. A fejlécben megtalálható gombok, mint pédául a *Logout*, átkerülnek szintén ebbe az új menübe.

2.3.3. Komponensek

A felhasználók által használható felület *stateless* és *stateful* komponensekből áll. A *stateless* komponensek nem tartalmaznak dinamikusan változó adatokat, és állapottal sem rendelkeznek. Ilyen komponens például a *Logo*, a *HeaderLinks* vagy a *Home*. A *stateful* alkotóelemek ezzel szemben össze vannak kapcsolva egy-egy *MobX* állapotmenedzser osztállyal (3.2), ezáltal vannak közvetlen kapcsolatban az üzleti logikával. Ezeken a komponenseken keresztül reagál a weboldal a belső változásokra és a felhasználó tevékenységére.

2.3.4. UI routing

A webalkalmazás két részre osztható, amelyeket két konténer szimbolizál. Az első a *HomeContainer* amely a kezdőoldat tartalmazza. A webalkalmazás kezdőlapja egy statikus oldal. Innen a *Login* gomb irányít át minket a */admin* cím alatt található *AdminContainer*-be. Itt az alapértelmezetten betöltődő oldal az a kihívások áttekintését szolgáló */admin/challenges*. Az oldaló menü segítségével eljuthatunk a kihívás ajánlások oldalára, a */admin/suggestions* cím alatt. Ha rendelkezünk adminisztrátor joggal, akkor léphetünk a felhasználókat kezelő, */admin/users* oldalra is.

3. Technológiák és eszközök

A munkánk során igyekeztünk mindíg a legmodernebb és szükségleteinknek legmegfelelőbb technológiákkal dolgozni. Ebben a fejezetben röviden bemutatjuk az általunk használt fő technológiákat amelyeket a szerver, a webalkalmazás és a mobilalkalmazás készítése során használtunk. A fejezet végén említésre kerülnek azok az eszközök, melyek nagyban segítették a gyors és hatékony munkát.

3.1. Szerveroldali technológiák

Node.js A sokak által kedvelt és gyakran használt JavaScript kezdetben csak a webes alkalmazások kliens oldalát hódította meg. Ennek a népszerű programozási nyelvnek a szerver oldalon való felhasználását valósítja meg a Node.js, amely egy Chrome V8 JavaScript motorra épülő hatékony, eseményvezérelt környezet . Képes több kapcsolat egyidejű kezelésére, és a szálakra bontott alkamazásokkal ellentétben holtpontoktól mentes. Az alkalmazások fejlesztését nagyban elősegíti a Node.js moduláris jellege, szoros kapcsolatot ápol az npm [21] csomagkezelővel, ami a világ egyik jelentős JavaScript szoftver-nyilvántartása. A mi esetünkben külön jelentősge volt annak, hogy mint szerver, mint kliens oldalon ugyanabban a nyelvben fejlesszünk, ezért is részesítettük előnyben a Node.js-t más hasonló célú megvalósításokkal szemben, mint például a Spring.

Express.js Noha a Node.js egy teljes JavaScript motort nyújt, különböző külső könyvtárakra van szükségünk ahhoz, hogy megfelelő HTTP webszervert és RESTful API szervert készítsünk a segítségével. Az Express.js [20] egy minimalista, szintén JavaScript-ben írt keretrendszer Node.js alkalmazásokhoz, amely rengeteg funkcionalitással bővíthető az npm csomagok segítségével. Ideális web- és mobilalkalmazások szervereként és API szerverek felépítésére.

MongoDB A MongoDB [6] egy nyílt forráskódú dokumentum alapú adatbázis, amely a szoftverek fejlesztési folyamatának és a skálázhatóságának megkönnyítésére lett létrehozva. NoSQL adatbázis lévén, a relációs adatbázis tábla alapú struktúrája helyett BSON (Binary JSON) típusú dinamikus sémákat használ. A BSON helytakarékosabb és könnyebben feldolgozható az idexelés miatt mint a JSON és tartalmaz plusz adattípusokat. Az általunk tárolt adatok sokszínű szerkezete indokolta egy dokumentum alapú adatbázis használatát, így döntöttünk a MongoDB mellett.

Mongoose ODM Szükségünk volt egy könnyen kezelhető kapcsolatra az adatbázisunk és a Node.js szerverünk között, amely egyszerűvé és hatékonnyá teszi az adataink elérését és ma-

nipulását, ezért döntöttünk a Mongoose [7] mellett. A Mongoose egy objektum modellező JavaScript könyvtár, ami hidat képez a Node.js és a MongoDB között, egyértelmű séma alapú megoldást nyújt az alkalmazás adatainak ábrázolására. Tartalmaz beépített típusellenőrzést, típuskonverziót és lekérdezésmanipulációt.

3.2. Web technólogiák

A webalkalmazásunk alapjatként szintén a Node.js (3.1) és az Express.js (3.1) szolgált. A továbbiakban bemutatjuk a webklienshez felhasznált specifikus technológiákat.

React.js A React vagy React.js [17] egy a Facebook által fejlesztett JavaScript könyvtár amely felhasználói felületek készítésére készült. Számunkra nagy előnyt jelentett hogy tiszta JavaScrip-en alapszik és, például az Angular.js-el ellentétben, nem igényel TypeScript-et ahhoz, hogy teljesen kihasználjuk a lehetőségeit. A legmérvadóbb érv, amikor a React és az Angular között döntenünk kellett, az volt, hogy a React nem használ külön sablonokat, hanem az újrahasznosítható komponenseket HTML és főleg JavaScript segítségével építi fel, ezzel ellentétben az Angular nyelv-specifikus sablonszintaxist használ (például az *ngIf* vagy *ngFor*), melyek elsajátítása külön energiát emészt fel a fejlesztés során.

A React legnagyobb újítása az hogy a lehetővé teszi nagyméretű oldalak megjelenítését és változtatását anélkül hogy újra kellene töltenie a felhasználónak az oldalt. Ezt a 'Virtual Document Object Model' vagy ' Virtual DOM' segítségével valósítja meg. A React készít egy adatstruktúrát a memóriában és mindíg újraszámolja a változásokat, majd hatékonyan frissíti a böngészőben megjelenő DOM-ot. Mindíg csak a háttérben levő adatoknak megfelelő komponensek jelennek meg az oldalon. Ez lehetővé teszi a fejlesztők számára, hogy úgy írják meg a kódot, mintha az egész oldal újratöltődne minden változtatáskor, hiszen a React könyvtárak csak azokat a komponenseket jelenítik meg újra, amelyek ténylegesen változtak. A React alapból támogatja az ES2015 másnéven ES6 JavaScript szabványt, amely egy sor újítást és javítást tartalmaz a korábbi standardokhoz képest. A használt alapszintaxis a JSX, amely megengedi HTML beágyazását a JavaScript kódba. Sok keretrendszer egy sablonnyelvet használ, amely megengedi a kód beszúrását a sablonokba. A React esetében ez fordítva történik, a JSX megengedi a sablon beékelését a kódba.

Webpack A fejlesztés során rengeteg különböző fájltípus és függőség került bele a webalkalmazásunkba és mindezeknek még tövábbi tranzitív függőségei voltak. Szükségünk volt egy olyan technológiára amelyel egy csomagba tudjuk mindezt tenni, és ezáltal egy függőségmentesen kiadható terméket kapjunk. Erre a problémára adott tökéletes megoldást a Webpack [18], amely egy statikus csomagkészítő modern JavaScript alkalmazásokhoz. Az alkalmazás kódjá-

nak feldolgozása során a Webpack felépít egy rekurzív függőségi gráfot, ami tartalmazza az összes modult amire az alkalmazásnak szüksége van. Ezután becsomagolja az összes modult egy vagy több csomagba, amik ezután használhatók önálló alkalmazásként.

Babel A React (3.2) kód irása során a JSX kiterjesztést használtunk, mert szükségünk volt a HTML elemek beszúrásának lehetőségére a komponenseik felépítése során. Ezt a kódot viszont a böngészők nem tudják értelmezni, ezért szükségünk volt egy fordítóra, amely az elkészült alkalmazást tiszta JavaScript-é fordítja.

A Babel [1] egy source-to-source kompilátor, tehát egy adott forráskódot fordít szintén forráskóddá. A 'babel-loader' kiegészítő segítségével a Webpack képes a megírt magasabb szintű és annotációkkal ellátott JavaScript kódot egy egyszerűbb, a böngésző által is feldolgozható formára hozni. Az sokszínű, bárki álta elérhető bővítő csomagok a Babel-hez biztosítják nem csak a JSX, hanem más kiterjesztésű állományok feldolgozását is. Mi a *babel-preset-react* csomagot használtuk a React kód fordítására, és a *css-loader* kiegészítőt a CSS fájlok feldolgozására.

MobX Az összes React komponens rendelkezik állapotokkal, hiszen ezen alapszik a React dinamikus jellege, ezért volt szükség egy állapot menedzsment rendszerre, mert rengeteg problémát okozott és okozhatott volna az állapotok inkozisztens változtatása. Erre a problémára nyujthatott megoldást a Redux illettve a MobX. Végül az utóbbi mellett döntöttünk egyszerúsége és célratörő volta miatt. A MobX egy egyszerű és könnyen skálázható állapot menedzsment rendszer. Habár külön csomagként is elérhető, a legtöbben a React.js-el együtt használják. A használatával elértük, hogy mindíg konzisztens adatokkal dolgozhatunk, ugyanaz az adathalmaz van az üzleti logika szintjén mint amit a felhasználó éppen lát a képernyőn. A MobX működése a *store* elemeken lapaszik, melyek úgy működnek mint egy nyilvántartás, ahol az adatok mindíg a legfrissebb állapotban vannak, és a köztük levő összefüggések alapján újraszámolódnak, ha változtatás történik.

Bootstrap A Bootstrap egy nyílt forráskódú kliens oldali dizájn keretrendszer, amely hozzáférést nyujt rengeteg előre megírt komponenshez, mintákhoz és dizájnelemhez, melyek segítségével egy *responsive* oldat rakhatunk össze. A *responsive* oldalal célja hogy megfelelő felhasználói élményt nyújtsanak bármilyen eszközön, a lehető legkevesebb kihasználatlan hellyel, görgetéssel és átméretezéssel. Az alap Bootstrap csomag mellett mi még használtuk a 'react-bootstrap' npm csomagot amely a gyakran használt komponensek React implementációját tartalmazza.

3.3. Mobil technólogiák

React Native A cross-platform mobilalkalmazások fejlesztésének kezdetekor, a fejlesztők egy webaplikációt készítettek, amelyet egy leegyszerüsített böngésző segítségével jelenítettek meg a felhasználók számára. Ennek a megközelítésnek a legfőbb hátránya a lassúság és a megbízhatatlanság mellett az volt, hogy a megszokott Android-os és iOS-es kinézet helyett egy teljesen más felület fogadta a felhasználókat. Ennek hatására jöttek létre azok a kezdeményezések, amelyek a cross-platform applikációknak natív, megszokott kinézetet adtak és nem egy böngészőben futottak. A választásunk a React Native-re [12] esett, amely egy olyan keretrendszer melyben natív mobilalkalmazásokat lehet fejleszteni React.js-t (3.2) illetve tiszta JavaScriptet használva. A Reacthoz hasonlóan a React Native is komponenseken alapszik, viszont a webes komponensek helyett natív komponenseket használ. A React Native npm-en keresztül sok webes könyvtár használatát engedi meg. Az összes JavaSricpt kód a végén egy natív alkalmazássá fordul át, szó sincs arról hogy egy emulátorban futna a JavaScript applikácó. Emiatt és a React dinamikus természete miatt a React Native alkalmazások gyorsak és nagy szabadságot adnak a fejlesztőknek.

Native Base A Native Base [10] egy ingyenes nyilt forráskódú UI komponenseket tartalmazó könyvtár React Native-hez. Segítségével az elkészült alkalmazás az adott készüléknek megfelelő natív kinézettel rendelkezik. Célunk a Native Base használatával az volt, hogy mind az Android, mind az iOS felhasználóknak egy megszokott és kedvelt kinézetű applikációt tudjuk nyújtani anélkül hogy natív kódot kellene használni. A Native Base folytatja a React Native és az Expo (3.4) által mutaott irányt és mentesít a natív kód használata alól.

3.4. Eszközök

Expo Az Expo.io [3] egy ingyenes, nyilt forráskódú eszközcsomag ami a React Native köré épült, megkönnyítve a natív iOS és Android appok készítését. Az Expo segítségével készült appok tartalmazzák az Expo SDK-t amely egy natív és JavaScrip kódot tartalmazó könyvtár. Az Expo SDK elérést biztosít a készülékek rendszer szintű funkcionalitásaihoz mint például a tárhely, kamera, névjegyzék, helymeghatározó szenzor és hasonló hárdverelemek. Ez azt jelenti hogy a mobilapplikáció fejlesztése során nincs szükség sem Xcode-ra sem Android Studióra, sőt natív kódot sem kell írni. Az Expo könnyű hozzáférést enged a népszerű szolgáltatásokhoz, melyek kezelése nehézkes lehet, mint például az értesítések kezelése és natív bináris fájlok készítése, melyeket fel lehet tölteni a különböző áruházakba.

Visual Studio Code Mivel a projekt fejlesztése során ügyeltünk, hogy ne kelljen natív kódot írni, a Visual Studio Code volt a legmegfelelőbb fejlesztői környezet. A kiegésziítők segítségével a projektünkhöz tudtuk szabni a felületet, egységesíteni tudtuk a kódolási konvenciókat,

melyektől az eltérést autómatikusan jelezte a szerkesztő. A beépített git kiensnek köszönhetően nem volt szükségünk külső verziókövető kliensre sem.

ESLint Az ESLint [19] egy nyilt forráskódú statikus kódellenőrzésre használt kiegészítő, amelyet gyakran használnak hogy hibás vagy a konvencióknak nem megfelelő kódot megtalálják akár nagyobb projektekben is. A JavaScript, mivel egy gyengén típusos nyelv, könnyedén kerülhet bele a fejlesztés során hiba, amit compilálás híjján csak futtatás során lehet megtalálni. A kódellenörző kiegészítők, mint például az ESLint segítenek megtalálni a hibákat anélkül hogy futtatni kelljen a kódot. Az előre definiált lintelési szabályok alapján dolgozik, amelyek kiegészíthetők és felülírhatók bővítőcsomagok vagy akár a fehasználók által is. Az ESLint Node.js segítségével íródott, hogy egy letisztult futtatási környezetet nyújtson és könnyen installálható legyen npm segítségével.

Mocha & Chai A Mocha [5] egy olyan funkciókban gazdag JavaScript teszt keretrendszer, amely Node.js vagy böngésző segítségével futtatható. Az aszinkron tesztelés lehetőségeit kihasználva a tesztek egyszerre futnak, miközben a hibákat a megfelelő tesztesetekhez társítja. A Chai [2] egy *Behaviour Driven Development / Test Driven Development* ellenörző könyvtár Node.js-hez és a böngészők számára, amely könnyedén párosítható bármilyen javascript tesztelési keretrendszerrel. A Mocha & Chai párost népszerűségük, egyszerüségük és hozzáférhetőségük miatt válaszottuk [29].

Jest A Jest [4] egy JavaScript alapú tesztelési megoldás React.js és React Native applikációkhoz. Párhúzamos tesztek futtatásával maximalizálja a teljesítményt, a konzolban megjelenő üzenetek egy bufferbe kerülnek majd a tesztek eredményeivel együtt jelennek meg. Elkülöníti a teszteseteket és nem engedi hogy egymás *state*-ét befolyásolják. Beépített lefedettség ellenörzéssel rendelkezik, amit konzol üzenet keretében jelenít meg. Mivel közös a fejlesztő, ezért a React appok alapból tartalmazzák a Jest idnításához szükséges feltételeket, csupán a teszteket kell megírni az erre kijelölt fájlba és már futtatható is a rendszer.

Gitlab és Docker A projekt verziókövetését GitLabon keresztül valósítottuk meg. Szétválasztottuk a webalkamazás, a szerver és a mobilalkalmazás forráskódját három különböző *repository*-ba az átláthatóság érdekében. Minden új funkcionalitás fejlesztése külön ágon történt, melyeket a *development* ágon egyesítettünk a kód átnézése után. A *master* ágra olyan verziók kerültek, melyek önállóan és teljes mértékben működőképesek.

A *Gitlab Continous Integration* fontos szerepet játszott a fejlesztési folyamatban, hiszen egy folyamatos képet kaphattunk az aktuális állapotról. Első lépésként a teszteket futtatuk le, majd ha ezek sikeresek voltak, akkor az ESLint segítségével vizsgáltuk át a forráskódot elírásokat

és hibákat keresve. Az API szerverünk *master* ágára kerülő kódot ki is telepítettük egy külső szerverre, hogy gyors elérést biztosítsunk a fejlesztés során, és ne kelljen lokálisan futtatni a szervert, ahhoz hogy a kliensek megfelelően működjenek.

A kitelepítést dockerizálás segítségével automatizáltuk, így amikor egy új verzió kerül fel a *master* ágára, elkészítünk egy új *docker image*-t és azt a megfelelő *edu.codespring.ro* szerverre telepítjük.

MongoDB Compass A MongoDB adatbázis használata szerves részét képezi a fejlesztési folyamatnak. Kliensként a hivatalos MongoDB Compasst használjuk, amely jelentősen megkönnyíti az adatok kézi manipulációját a konzolos hozzáféréssel szemben.

4. A kliensalkalmazások működése

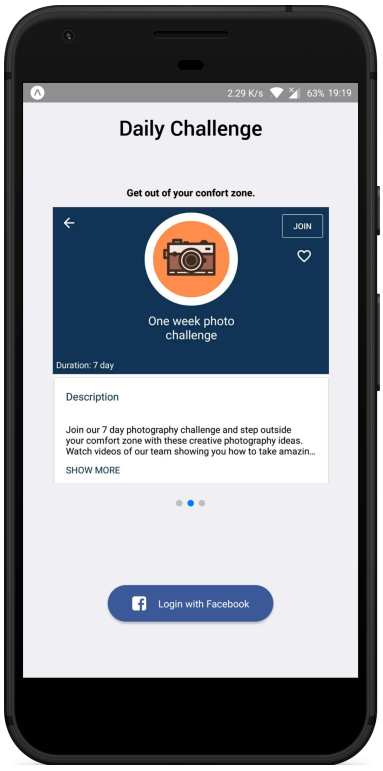
Ez a fejezet a mobilalkalmazás és a webes felület működését valamint lényegesebb funkcionalitásainak használatát ismerteti egy rövid leírás és különböző helyzetekből származó képernyőképek segítségével.

4.1. A mobilalkalmazás használata

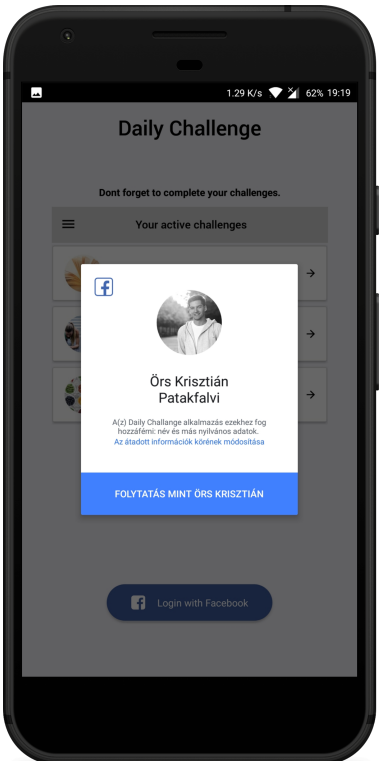
Az alkalmazás megnyitása után az 13. ábrán látható bejelentkezésre szolgáló nézet fogadja a felhasználót. Itt egy képsorozat nyújt betekintést az alkalmazás néhány funkcionalitásába. Lentebb egy a Facebookkal való bejelentkezést biztosító gomb kapott helyet. Ennek segítségével lehet bejelentkezni a Facebook fiókunkba és megadni az alkalmazás által kért információkat a profilunkról. Ha a felhasználó Android-ot futtató telefont használ és telepítve van a Facebook alkalmazása, akkor a 14. ábrán látható felugró ablakban fogja a jogokat kérni az alkalmazás.

Sikeres bejelentkezést követően az aktív kihívások áttekintésére szolgáló nézet jelenik meg. Első bejelentkezésnél még a felhasználónak nincsenek kihívásai, így az alkalmazás ajánlja is, hogy csatlakozzon egybe.

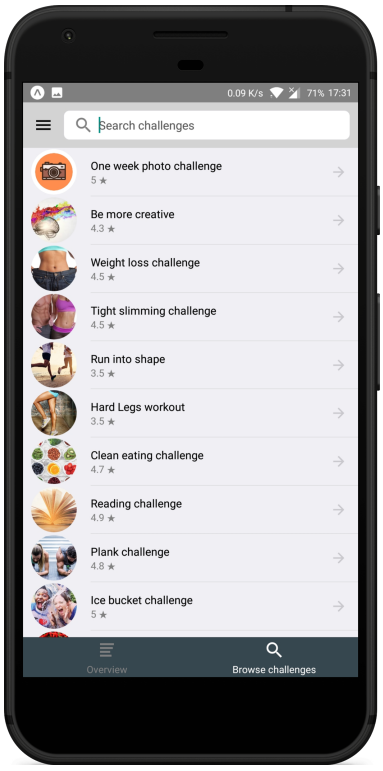
A képernyő alján két fül látható, az első az előző bekezdésben említett áttekintést takarja, a másik pedig az elérhető kihívások listájához vezet. Ha a másodikra rákattint a felhasználó vagy



13. ábra. Bejelentkezési nézet képsorozattal, amely betekintést nyújt az alkalmazásba

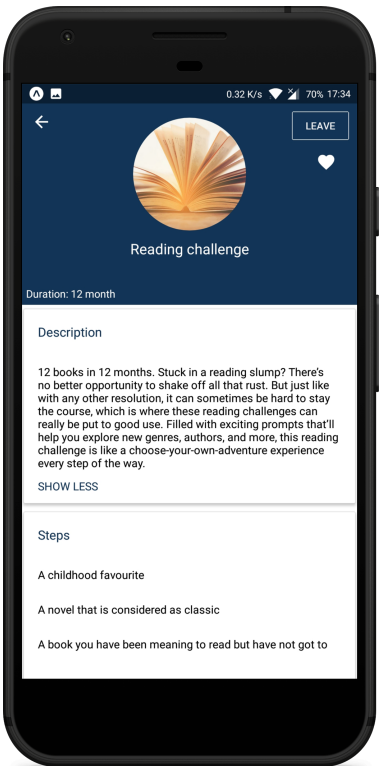


14. ábra. A szükséges hozzáférések megadása a felhasználó adataihoz

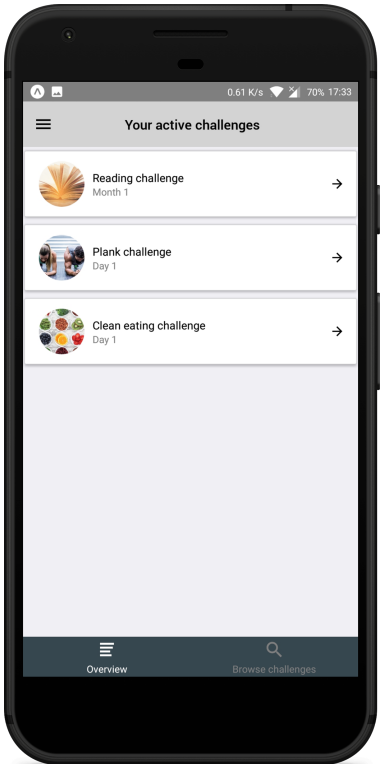


15. ábra. Böngészés a mindenki számára elérhető kihívások listájában

16. ábra. Van lehetőség keresni a kihívások között



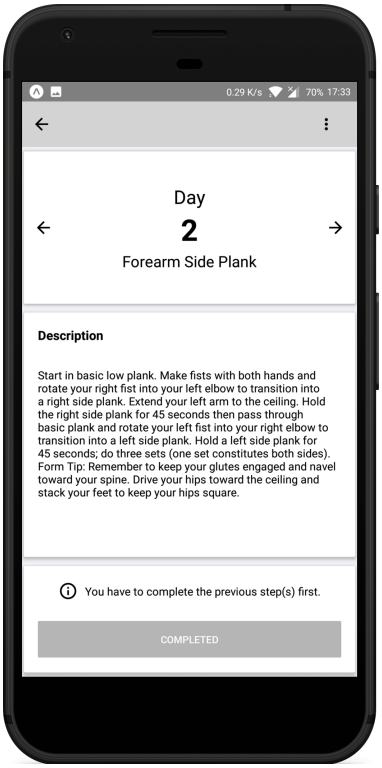
17. ábra. Megtekinthetjük egy kiválasztott kihívás részleteit



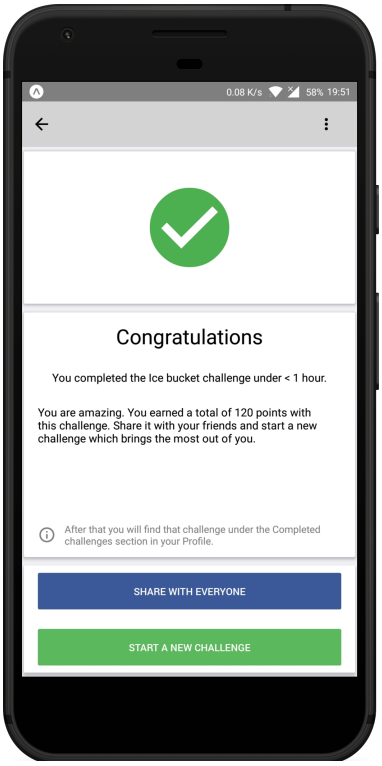
18. ábra. Aktív kihívások áttekintését szolgáló nézet

egyszerűen jobbra húzza az ujját a kijelzőn, akkor egyből a 15. ábrán látható nézethez jut. Itt a szerveren elérhető publikus kihívások kerülnek kilistázásra. Mindegyiknek megjelenik a képe, a címe és az értékelése 1-től 5-ig terjedő skálán. Ahogy a 16. ábra is mutatja ezek között keresni is lehet a képernyő tetején látható keresősávba gépelve. Három karakter beírása után elindul a keresés és csak azok a kihívások jelennek meg, amelyeknek a címében vagy leírásában szerepel a kért szöveg. Itt bármelyik elemre kattintva a kiválasztott kihívás részleteit tekintheti meg a felhasználó egy új, a 17. ábrához hasonló nézetben. Itt egy részletesebb leírást lehet olvasni, bepillantást kaphat a felhasználó a kihívás lépéseibe valamint más hasznos információkat tudhat meg. Ugyanitt a jobb felső sarokban lévő *Join* vagy *Leave* gombokkal lehet csatlakozni illetve kiszállni egy kihívásból. Ha egy általunk már befejezett kihívást nézünk épp, akkor itt egy nem kattintható *Done* feliratú gomb jelenik meg, jelezve, hogy teljesítettük már a kiválasztott kihívást. Ez alatt egy szív alakú gomb látható, amelynek ha csak a körvonala látható, akkor a kihívás nincs a felhasználó kedvencei között, de kattintással hozzájuk adható, illetve ha kitöltött szív látható, akkor már a kedvencek között van, de kattintással eltávolítható onnan.

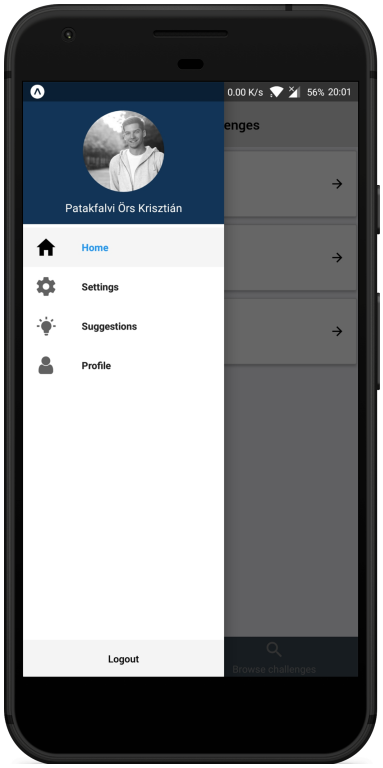
Hogyha a felhasználó csatlakozott legalább egy kihíváshoz, akkor a már említett aktív kihívásokat tartalmazó nézetben fognak ezek megjelenni, a 18. ábrán látható módon. Itt láthatja a felhasználó, hogy melyik kihívásában milyen szinten áll, hányadik lépés következik. Bármelyik elemre kattintva juthat el a lépések részletesebb áttekintéséhez, amit a 19. ábrán is látható.



19. ábra. Aktív kihívások lépéseit és azok leírásait szolgáltató nézet



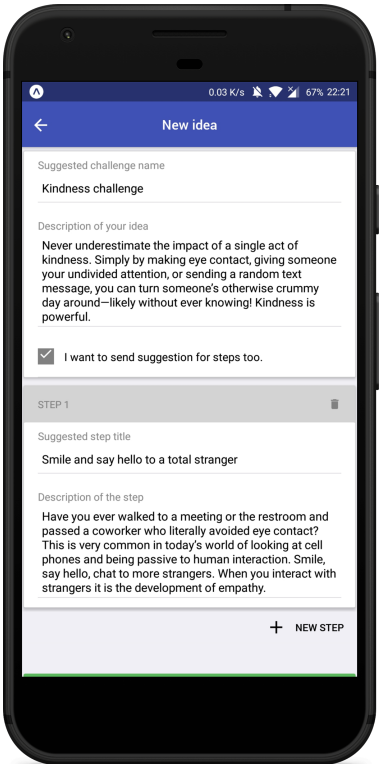
20. ábra. Gratulációs ablak mely kihívás befejezése esetén jelenik meg



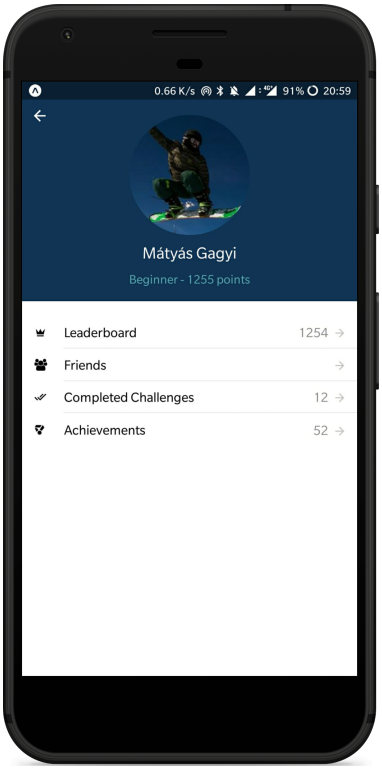
21. ábra. Az alkalmazás egységei közötti navigálásra szolgáló oldalsáv

Itt részletes leírást és utasításokat találhat a lépésekről, megnézheti az előző és következő lépéseket is, valamint teljesítettnek jelölheti a soron következő lépést. A *Completed* gomb felett (amellyel befejezettnek jelölhetünk egy lépést) látható információ arról is, hogy a kiválasztott lépés milyen állapotban van. Lehet, hogy olyan lépést nézünk, aminek a teljesítéséhez még előző lépéseket kell elvégezzünk, vagy épp már elvégeztük azt. De ha a megfelelő lépésen vagyunk, akkor is kapunk információt arról, ha még várnunk kell, mert túl hamar akarjuk teljesítettnek jelölni a lépést, vagy épp már késésben vagyunk. Ha a felhasználó befejezi az utolsó lépését is a kihívásnak, akkor a 20. ábrán látható gratulációs nézetet fogja látni, ahol információkat kap arról, hogy mennyi idő alatt sikerült végezzen a kihívással, illetve mennyi pontot szerzett vele összesen. Ugyanitt lehetőség nyílik a teljesítmény megosztására is a közösségi hálókon vagy más szolgáltatásokon keresztül.

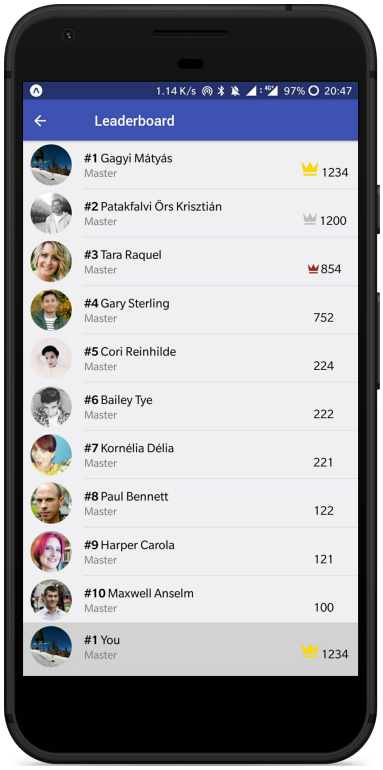
Az alkalmazáson belül a különböző nagyobb egységek közötti navigálásra a 21. ábrán lévő oldalsáv használható. Ezt az oldalsávot a több helyen is a fejlécben megtalálható menü ikon megérintésével vagy a képernyő bal szélétől befele simítva csalogathatjuk elő. Az eddigi bekezdésekben leírt nézetek és funkcionalitások mind a *Home* nevű menüpont alatt érhetőek el. Ennek a menüpontnak a megérintése navigál az elején bemutatott aktív kihívások áttekintését szolgáló nézethez. Még további három menüpontot és egy az alkalmazásból való kijelentkezésre haszálható *Logout* gombot találhatjuk itt.



22. ábra. Ötlet létrehozására és beküldésére szolgáló nézet



23. ábra. A felhasználó saját profilja



24. ábra. A felhasználó által ajánlott kihívás ötletek

A következő *Settings* menüpont az elérhető alkalmazásbeállításokat gyűjti össze egy listában. Két beállítási lehetőséget találunk itt, az elsőnél a központi szerver elérési útvonalát lehet módosítani, esetleges szervermigráció vagy más okok esetén. A második listaelem az értesítések beállításáért felelős nézethez navigál. Itt még a nézet megnyitása előtt információt látunk arról, hogy az alkalmazásban globálisan az értesítések ki vagy be vannak épp kapcsolva. Ha megnyitjuk a nézetet, akkor pontosabban beállíthatjuk, hogy mire vonatkozóan szeretnénk értesítéseket kapni.

A 21. ábrán a következő menüpont a *Suggestions* nevet viseli, itt lehet új kihívás ötleteket ajánlani a tartalommenedzserek fele. Megnyitva a menüt, egy listában tekintheti meg a felhasználó azokat az ötleteket, amiket ő ajánlott be. Az ötletek az állapotuk alapján vannak csoportosítva, ami 4 féle lehet. *Suggested*, ha a felhasználó beküldte az ötletét, de még senki nem nézte át, nem jutott még sorra. Ekkor a beküldött ötlet még szerkeszthető vagy törölhető. A következő állapot *accepted* vagy *rejected* lehet, annak függvényében, hogy a tartalommenedzser, aki átnézte az ötletet lát-e benne elég fantáziát ahhoz, hogy egy teljes értékű kihívást alakítsanak ki belőle. Innentől kezdve a ötletek már nem szerkeszthetőek és a kettő közül csak a rejcted, azaz visszautasított ötleteket van lehetősége a felhasználónak törölni. Ha egy ötlet elfogadásra került akkor a tartalommenedzserek elkezdenek dolgozni az ötlet kibővítésén és a kihívás véges állapotának elkészítésén. Az utolsó lépés a *published* állapot, ami azt jelzi, hogy a beküldött ötletet kidolgozták és a kész kihívást elérhetővé tették az összes felhasználó számára, tehát

megtalálható már a kihívások listájában és csatlakozni lehet hozzá.

A listában szereplő ötletekre kattintva nézhetők meg a részletek, valamint ott jelennek meg a szerkesztésre vagy törlésre szolgáló lehetőségek is. Ezeken kívül a publikált állapotú kihívásoknál egy gomb is megjelenik, ami egyből a közzétett kihívás részleteihez visz, ahol csatlakozni is lehet. A jobb felső sarokban megfigyelhető még egy hozzáadás gomb is, ami segítségével új ötleteket lehet létrehozni és beküldeni. Ha rákattint a felhasználó, akkor a 22. ábrán látható nézethez jut, ahol megadhatja az ötlete részleteit. Mindenképp szükséges egy cím és egy leírás a javaslathoz, de ezen kívül lehetőség van küldeni ötleteket a lépésekkel kapcsolatban is. Ehhez be kell pipálni az *I want to send suggestions for steps too* jelölőnégyzetet, és ezzel elérhetővé válik a *New step* gomb. A hozzáadott lépéseknél is kötelező megadni a címet valamint egy leírást mellé. Ha kész vagyunk, akkor a *Send the idea* gombbal továbbíthatjuk a javaslatunkat.

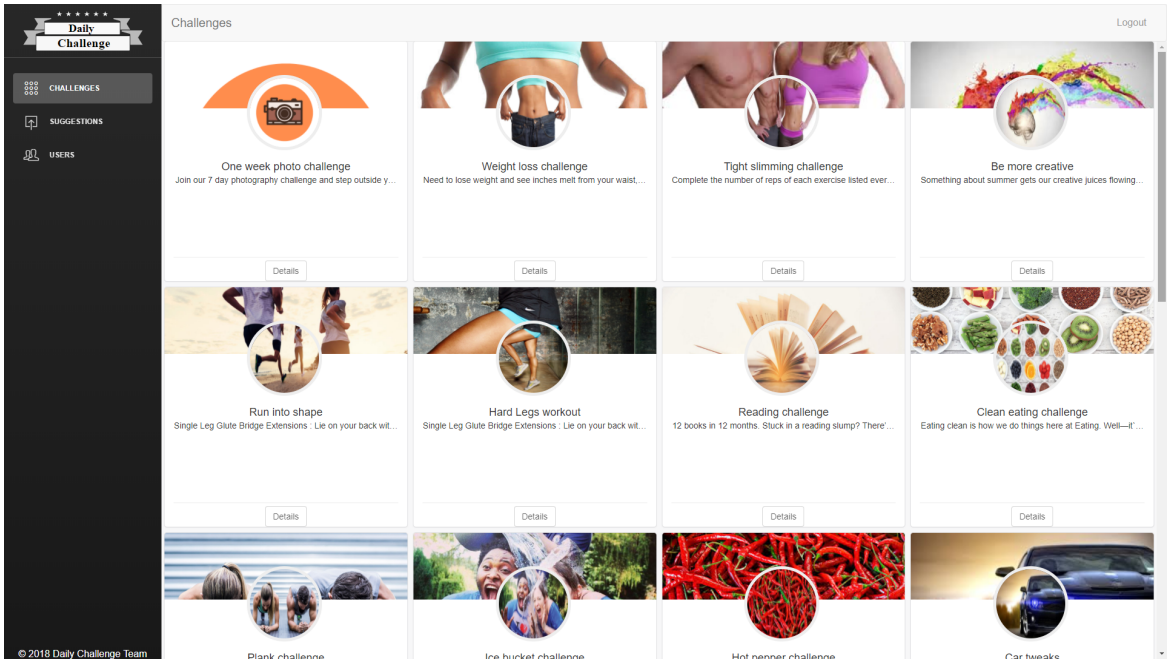
Az oldalsávon az utolsó menü a *Profile* nevet viseli és ez navigál át a felhasználó profiljához, amit a 23. ábrán láthatunk. Itt a fejlécben a képét, nevét és összpontszámát láthatja a felhasználó. Lentebb egy gomb látható *Leaderboard* felirattal, ez visz az 24. ábrán látható raglistát tartalmazó nézethez. Itt leolvasható, hogy globálisan milyen helyet foglal el a felhasználó az előbb említett pontszáma alapján. Visszatérve a profil oldalra a ranglista gomb alatt két fül található, az első fül a felhasználó befejezett kihívásait tartalmazza egy listában, ahol a listaelemek kattintható. A második fül a továbbfejlesztések során lesz használva a kitüntetések listázására.

4.2. A webes felület használata

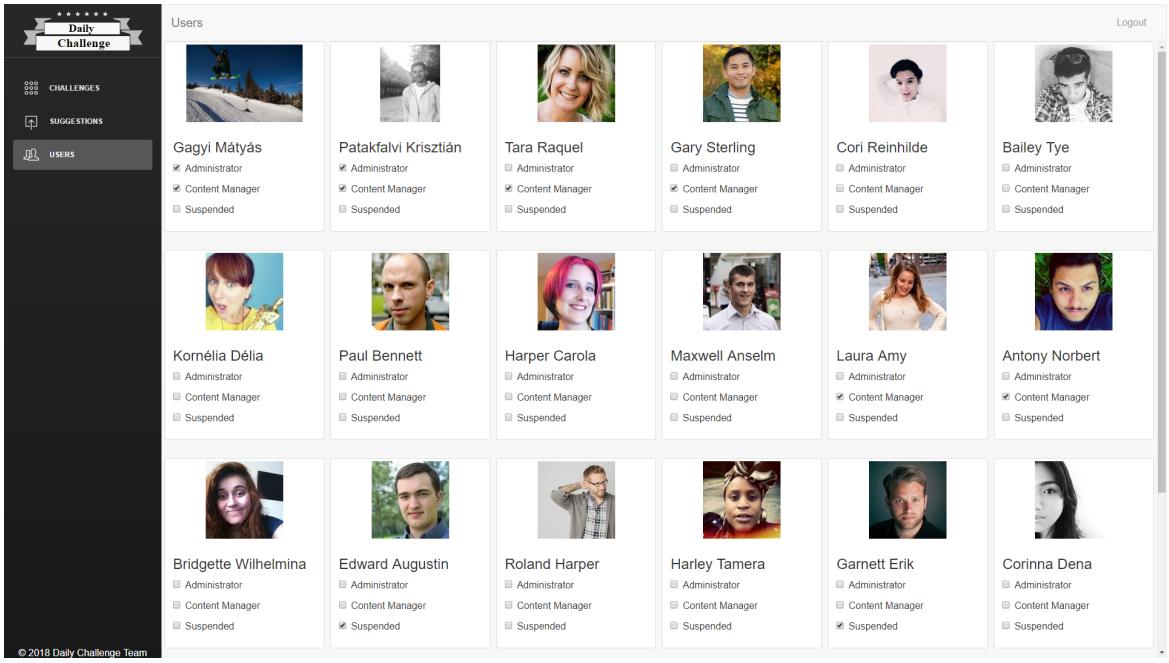
A webes alkalmazásunk fő szolgáltatásait csak a tartalommenedzser vagy adminisztrátor jogkörrel rendelkező felhasználók vehetik igénybe, de bárki megtekinheti a főoldalt, ahol megjelenik az alkalmazás rövid leírása, a fejlesztőcsapat elérhetősége és kész mobil applikáció letöltési lehetőségei az Apple Store-ban illetve a Google Play Áruházban. Innen a felhasználó a bejelentkezés gomb használatával tud Facebook segítségével bejelentkezni és továbblépni. Amennyiben a felhasználó rendelkezik tartalommenedzser vagy adminisztrátor jogkörrel, bejelentkezés után a Vezérlőpult jelenik meg. Ha nincs felhatalmazása a felhasználónak akkor egy erről informáló üzenet kíséretében visszatér a Kezdőoldalra.

A Vezérlőpult felületen a felhasználóknak lehetőségük van a Kihivások kiliztázására és kezelésére (25. ábra). Az oldalsó menüben kiválasztva a *Suggestions* fület lehetőségük nyílik a felhasználók álltal beküldött javaslatok áttekintésére. Rákattintva egy javaslatra, annak adatai egy szerkesztető felületen jelennek meg, ahol változtatásokat lehet végrehajtani az ajánláson, lehet annak kategóriáját változtatni és akár publikálni is, mint teljes értékű kihívás. Az ajánlásokal kategoriánként lehet kilistázni, a kategóriák között a névvel ellátott fülek segítségével lehet váltani.

Az *Admin* jogkörrel rendelkező felhasználók számára megjelenik az oldalsó sávban egy *Users* menüpont. Erre a menüpontra kattintva megjelenik a felhasználók kezelésével foglalkozó oldal, ahol lehetőség van a felhasználók között keresni, kilistázni őket és a jogköreiket megváltoztatni (26. ábra).



25. ábra. A kihívásokat áttekintő webes felület



26. ábra. A felhasználókat áttekintő webes felület

5. Következtetések és továbbfejlesztési lehetőségek

Jelen dolgozat bemutatta a sikeresen megvalósított Daily Challenge mobilalkalmazás és a hozzá tartozó webes felület valamint API szerver architektúráját, a felhasználók által elérhető funkcionalitásokat és a felhasznált technológiákat. Az előre kitűzött céloknak megfelelően, az alkalmazás egy helyen, egy egységes felületen teszi elérhetővé felhasználói számára a kihívásokat, a webes felület pedig biztosítja a szükséges eszközöket a háttérben tevékenykedő személyeknek, hogy karbantarthassák és felügyelhesség az alkamazásban megjelenő tartalmakat.

Az alkalmazás és a webes felület későbbi fejlesztésének főbb prioritásai közé tartozik a felhasználók közötti kapcsolatteremtés és komunikáció. Lehetőséget fogunk teremteni a felhasználóknak, hogy hozzászólásokat fűzhessenek egyes kihívásokhoz, egyes lépésekhez, befejezés után értékelhessék azokat, ezzel jelezve a többi érdeklődő fele, mennyire találták érdekesnek és hasznosnak a kihívást. Felhasználásra fog kerülni a Facebook által kínált barátlistát is az alkalmazásban, ezáltal mindenki külön követhetni tudja az ismerősei teljesítményét, érdeklődéskörét és akár ki is hívhatja őket. Lehetőséget szeretnénk teremteni chatelésre, akár személyek között, akár kihívások keretében, ezzel elősegítve a kommunikációt a felhasználók között. A ranglista kibővítése is szerepel a terveink között, melynek célja az lenne, hogy ösztönözze a felhasználókat minél több kihívás teljesítésére. Elérhető kitünktetésekkel tervezzük érdekesebbé tenni a kihívások teljesítését, hogy maradandó értékkel tudjuk jutalmazni a kitartó felhasználókat.

Szeretnénk bevezetni egy új szerepkört, melynek a hozzászólások moderálása lenne a feladata, munkáját egy automatizált szűrő segítségével végezné. Emellett szeretnénk opciókat adni az Adminoknak hogy felfüggeszthessenek felhasználókat illetve megvonhassanak különböző tartalmakat tőlük rossz magaviselet esetén.

A kíhívások készítését és ajánlását szeretnénk átdolgozni, felépíteni egy előre definiált használható nap-típusokból álló csomagot, amelyből könnyedén lehet egy kihívást összerakni. Ezekhez a naptípusokhoz felhasználnánk a telefon érzékelői által adott lehetőségeket, mint például a helymeghatározást, giroszkópot, kamerát és mikrofont.

Szükségét érezzük egy offline mód elkészítésének is, aminek köszönhetően internet kapcsolat nélül is lehetne a kihívásokkal haladni, majd később a kapcsolat helyreállásakor ezeket szinkroninzálni.

A Facebookos bejelentkezés mellett szeretnénk csatolni az alkalmazásunkat más platformokhoz is, ezáltal lehetőséget adva a bejelentkezésre akár Google azonosítóval vagy Linkedln profillal.

Hivatkozások

[1] *A Babel hivatalos dokumentációja*. URL: https://babeljs.io/ (utolsó elérés dátuma: 2018. ápr. 03.)

[2] *A Chai hivatalos dokumentációja*. URL: http://www.chaijs.com/ (utolsó elérés dátuma: 2018. márc. 13.)

[3] *A Expo.io hivatalos dokumentációja*. URL: https://expo.io/ (utolsó elérés dátuma: 2018. ápr. 06.)

[4] *A Jest hivatalos dokumentációja*. URL: https://facebook.github.io/jest/ (utolsó elérés dátuma: 2018. ápr. 06.)

[5] *A Mocha hivatalos dokumentációja*. URL: https://mochajs.org/ (utolsó elérés dátuma: 2018. márc. 13.)

[6] *A MongoDB hivatalos dokumentációja*. URL: https://docs.mongodb.com/manual/ (utolsó elérés dátuma: 2018. márc. 13.)

[7] *A Mongoose hivatalos dokumentációja*. URL: http://mongoosejs.com/ (utolsó elérés dátuma: 2018. márc. 13.)

[8] *A Mozilla hivatalos dokumentációja*. URL: https://www.mozilla.org/hu/about/ (utolsó elérés dátuma: 2018. ápr. 01.)

[9] *A NativeBase - Components hivatalos dokumentációja*. URL: http://docs.nativebase.io/Components.html#Components (utolsó elérés dátuma: 2018. márc. 13.)

[10] *A NativeBase hivatalos dokumentációja*. URL: https://docs.nativebase.io (utolsó elérés dátuma: 2018. márc. 13.)

[11] *A React Native Components and APIs hivatalos dokumentációja*. URL: https://facebook.github.io/react-native/docs/components-and-apis.html (utolsó elérés dátuma: 2018. márc. 13.)

[12] *A React Native hivatalos dokumentációja*. URL: https://facebook.github.io/react-native/docs/tutorial.html (utolsó elérés dátuma: 2018. ápr. 05.)

[13] *A React Native StyleSheet hivatalos dokumentációja*. URL: https://facebook.github.io/react-native/docs/stylesheet.html (utolsó elérés dátuma: 2018. ápr. 20.)

[14] *A React Navigation API hivatalos dokumentációja*. URL: https://reactnavigation.org/docs/api-reference.html (utolsó elérés dátuma: 2018. ápr. 20.)

[15] *A React Navigation hivatalos dokumentációja*. URL: https://reactnavigation.org/docs/getting-started.html (utolsó elérés dátuma: 2018. ápr. 20.)

[16] *A React.Component hivatalos dokumentációja*. URL: https://reactjs.org/docs/react-component.html (utolsó elérés dátuma: 2018. márc. 13.)

[17] *A React.js hivatalos dokumentációja*. URL: https://reactjs.org/tutorial/tutorial.html#what-is-react (utolsó elérés dátuma: 2018. ápr. 05.)

[18] *A Webpack hivatalos dokumentációja*. URL: https://webpack.js.org/concepts/ (utolsó elérés dátuma: 2018. ápr. 03.)

[19] *Az ESLint hivatalos dokumentációja*. URL: https://eslint.org/docs/about/ (utolsó elérés dátuma: 2018. ápr. 06.)

[20] *Az Express.js hivatalos dokumentációja*. URL: https://expressjs.com/ (utolsó elérés dátuma: 2018. ápr. 03.)

[21] *Az npm hivatalos dokumentációja*. URL: https://docs.npmjs.com/getting-started/what-is-npm (utolsó elérés dátuma: 2018. márc. 13.)

[22] *Callback function - Glossary | MDN*. URL: https://developer.mozilla.org/en-US/docs/Glossary/Callback_function (utolsó elérés dátuma: 2018. ápr. 01.)

[23] *Design Patterns - MVC Pattern*. URL: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm (utolsó elérés dátuma: 2018. ápr. 15.)

[24] *Error - JavaScript | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Error (utolsó elérés dátuma: 2018. márc. 24.)

[25] *JavaScript - Getter*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/get (utolsó elérés dátuma: 2018. márc. 30.)

[26] *JSON - JavaScript Object Notation*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON (utolsó elérés dátuma: 2018. márc. 24.)

[27] *Promise - JavaScript | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise (utolsó elérés dátuma: 2018. márc. 24.)

[28] *React Native - Networking - Using Fetch*. URL: https://facebook.github.io/react-native/docs/network.html#using-fetch (utolsó elérés dátuma: 2018. márc. 24.)

[29] *Testing Node.js with Mocha and Chai*. 2015. URL: http://mherman.org/blog/2015/09/10/testing-node-js-with-mocha-and-chai/#.WsNElohuaUk (utolsó elérés dátuma: 2018. ápr. 03.)

[30] *Understanding CORS*. 2018. URL: https://medium.com/@baphemot/understanding-cors-18ad6b478e2b (utolsó elérés dátuma: 2018. ápr. 18.)

[31] *What is NoSQL?* URL: https://www.mongodb.com/nosql-explained (utolsó elérés dátuma: 2018. ápr. 13.)

[32] *XMLHttpRequest - Web APIs | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest (utolsó elérés dátuma: 2018. ápr. 01.)