ETDK-dolgozat

Domokos Nikolette-Beatrice

Szimma Hunor

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

Informatika II.: innovatív számítástechnikai termékek, alkalmazások szekció

Kolozsvár, 2023. május 18–21.

Wanna Dance?

Latin táncfesztiválok szervezését és követését biztosító szoftverrendszer

Szerzők:

Domokos Nikolette-Beatrice

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

Szimma Hunor

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

Témavezetők:

dr. Sulyok Csaba, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Bartha Vivien, szoftverfejlesztő,

Codespring

Vad Bertalan, szoftverfejlesztő,

Codespring

CODESPRING

OUR CODE YOUR SUCCESS

Kivonat

Az elmúlt évtized során a latin táncfesztiválok és kongresszusok népszerűsége rendkívüli növekedésen esett át, a résztvevők száma gyakran több ezer személy fölé is emelkedhetett. Az ilyen rendezvények több napos programokat biztosítanak a jelenlévők számára, mint például különböző oktató jellegű workshopok, szociális programok, versenyek és természetesen hajnalba átnyúló bulik. Mindezen események szervezése a rendezők számára és azok nyomonkövetése a résztvevőnek egy kihívás lehet. Általános esetekben erre megoldást nyújthat különböző alkalmazások és szolgáltatások összehangolása.

A „Wanna Dance?” projekt célja, hogy az ilyen rendezvények lebonyolítását gördülékenyebbé tegye, valamint hogy a résztvevők élményének színvonalát növelje. Főbb funkcionalitások közé tartoznak az információk felvezetése és karbantartása a szervezők számára, az események követése a résztvevők számára, valamint a regisztrált felhasználók közötti szocializálódást elősegítő nyitott üzenetek és táncfelkérések küldése.

A szoftverrendszer rendelkezik egy mikroszerviz architektúrán alapuló szerverrel, valamint egy cross-platform mobilalkalmazással, amely elérhető Android és iOS eszközökön.

A dolgozat ismerteti a rendszer funkcionalitásait, architektúráját és megvalósítási módját, valamint a fejlesztésben felhasznált technológiákat és a továbbfejlesztési lehetőségeket.

Tartalomjegyzék

Bevezető

Amikor a tánc kerül szóba, legtöbben a profi táncosokra gondolnak, kifinomult mozdulatokra, és jól begyakorolt koreográfiákra. Sokan nem is mernek táncolni egy-egy eseményen, hiszen azt gondolják, hogy zavarba ejtő, ha nem profi szinten művelik. Pedig a tánc elsősorban nem a versenyzés miatt előnyös, hanem számos egyéb pozitív hatása van. A táncot, nem véletlenül, mozgó meditációnak is szokták nevezni [15]. A tánc emeli a szellemet, felszabadítja a testet, és harmonizálja a lelket, egy tökéletes módja ha az ember kikapcsolódásra, illetve mély élményekre vágyik. Rengeteg helyen adódik esély táncolásra, akár helyi, kisebb összejöveteleken, esküvőkön, szórakozóhelyeken, vagy akár a tánc köré épülő rendezvényeken, fesztiválokon. A táncfesztiválok körében egyre nagyobb népszerűségnek örvendenek a karibi stílusú táncok, mint a salsa, bachata, merengue, kizomba, zouk és ezeknek számos különböző alfajai.

Az ilyen latin táncokra jellemző a könnyedség, játékosság és az alapok egyszerű elsajátítása, így a hagyományos néptáncokkal ellentétben kevésbé szigorúak, kötöttek. Ennek köszönhetően nincs szükség egy előre betanult koreográfiára, hogy a táncosok bárkivel tudjanak táncolni, mivel a figurák improvizált módon vannak kitalálva és levezetve, így ennek köszönhetően a résztvevők szabadon cserélgethetik a partnereiket zenéről zenére. Ezekre a táncstílusokra alapuló fesztiválokra jellemző, hogy nem csak szórakozni lehet esténként, hanem tánctudást gyarapítani a napközben tartott különböző workshopokon.

Az ilyen jellegű táncfesztiválok, a fentebb említett tulajdonságuk miatt, egyre nagyobb népszerűségnek örvendenek. Manapság akár több ezer érdeklődőt is bevonzanak, akik részesei szeretnének lenni a fesztiválok által nyújtott élménynek. Annak érdekében, hogy a táncosok könnyebben lépést tudjanak tartani a történésekkel és értesüljenek a fontos információkról, illetve a szervezők hatékonyan tudják ezeket biztosítani, hasznos lenne egy teljes körű szoftverrendszer.

A *Wanna Dance?* projekt pontosan erre nyújt megoldást. Egy mobilalkalmazás keretein belül lehetőséget nyújt a szervezők számára, hogy egy egységes helyen tudják közzétenni a fontos információkat, és hatékonyan tudják értesíteni a résztvevőket a felmerülő módosításokról. Ezek az információk, stílusok, termek leírásai megfelelően összeszervezve megtalálhatóak egy helyen, ez által is könnyed betekintést nyújtva az adott fesztiválba. A résztvevők táncfelkéréseket tudnak intézni egymás között, ezzel megsokszorozva a lehetőségeiket. Mindemellett a beépített menetrendet követve a fesztivál teljes programjával

lépést tudnak tartani.

Léteznek különböző megvalósítások fesztiválok menedzselésének megkönnyítésére. Ilyen például a FestivApp[1] [6], ami több fesztivál programjába nyújt párhuzamosan betekintést az érdeklődők számára. Latin táncfesztiválok esetén ez a megoldás nem feltétlenül elégséges, ugyanis a *Wanna Dance?* táncfelkérési lehetőséget is biztosít a résztvevő táncosok számára. Az egyszerű információk megosztása mellett az applikáció lehetőséget nyújt órarend eltolásra, akár egy esemény, vagy akár több egymást követő eseménynél propagáció által, amelyekről a résztvevők értesítést is kapnak. Ezek elősegítik a csúszások menedzselését, így az órarend időben valós reprezentációt mutat mindenkor, ami segít a résztvevőknek megtervezni a workshoppok közti szüneteket és jobban beosztani az idejüket.

A projekt eredetileg egy disszertációs dolgozat alapjául szolgáló szoftverrendszerként indult, amit a Codespring Mentorprogram[2] egyik jelenlegi mentora, Vad Bertalan kezdett el fejleszteni. Az alkalmazás legfőbb célja az említett szociális aspektus megvalósítása volt, ez később, 2022 nyarán, a Codespring cég által szervezett gyakorlat, illetve az egyetemi csoportos tantárgy keretén belül folytatódott. A nyári gyakorlat folyamán e dolgozat szerzői, majd a csoportos projekt egyetemi tantárgy által Adorjáni Bíborka, Karda Brigitta, Boros Róbert Árpád és Krecht Ábel vitte tovább a fejlesztést.

A dolgozat a projektet, a fejlesztési folyamatot illetve a felhasznált eszközöket és technológiákat mutatja be. Elsősorban az alkalmazásban fellelhető két szerepkört és azok funkcionalitásait mutatja be (1. fejezet). Ezt követően a rendszer architektúrájára tér ki, mind a mobilalkalmazás és a szerver szempontjából (2. fejezet).. Majd bemutatja a fejlesztés alatt használt eszközöket és technológiákat (3. fejezet). Majd a részletesen betekintést nyújt az alkalmazás működésébe, táncos és szervező szemszögből is (4. fejezet) A dolgozat néhány következtetéssel és a továbbfejlesztési lehetőségek leírásával zárul.

[1] FestivApp - Általános szoftverrendszer rendezvények programjának böngészésére és menedzsmentjére (`https://festivapp.eu`)

[2] Codespring - `https://edu.codespring.ro`

1. Szerepkörök és funkcionalitások

Ez a fejezet nyújt betekintést a *Wanna Dance?* rendszer két szerepkörére, a *táncosra* és a *szervezőre*, valamint az ezek által elérhető funkcionalitásokra. Az alkalmazás csakis bejelentkezett felhasználók számára elérhető.

1.1. Táncos

Az alkalmazás alap szerepköre a *táncos* szerepkör, ez rendelkezik a legkevesebb jogosultsággal. A *táncos* felhasználónak lehetőségében áll a rendezvény információit megtekinteni, eseményeket böngészni és szűrni azokat napok szerint, illetve üzeneteket és táncfelkéréseket küldeni más résztvevők számára.

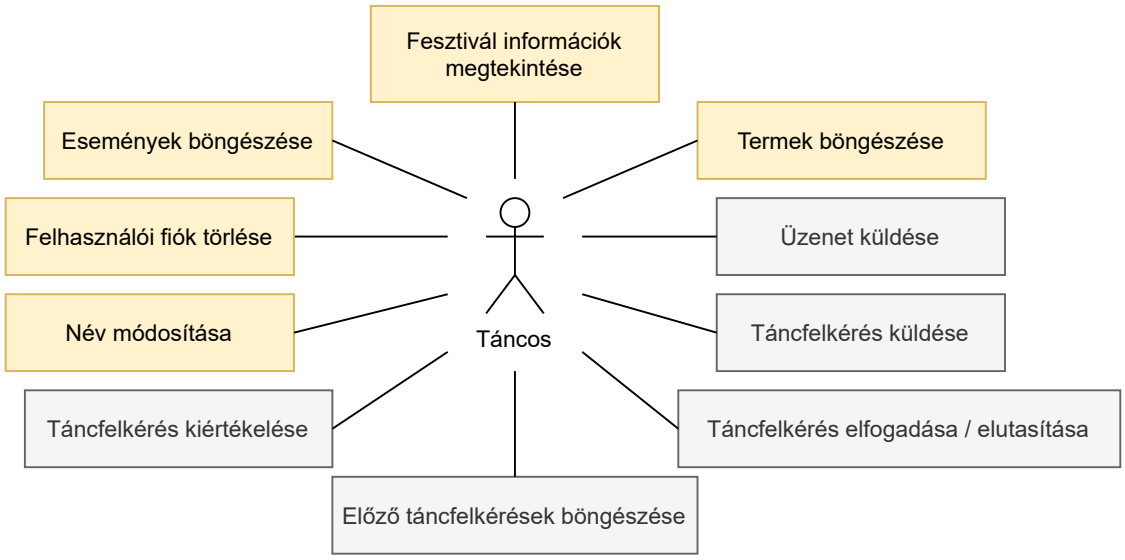
A fesztivál információi, mint annak neve, leírása, kezdeti- és végidőpontjai, tánc stílusai és rendelkezésre álló termei mind megtekinthetőek a táncos által. A felhasználók a termek bővebb leírásához, illetve az ezekhez rendelt eseménylistához is hozzáférhetnek.

Az események átlátható böngészésének érdekében, két típusú órarend áll a *táncos* szerepkörrel rendelkező felhasználó rendelkezésére. Az első nézet a fesztiválhoz tartozó összes eseményt adja vissza, termekként csoportosítva és időrendi sorrendbe rendezve. Abban az esetben, ha több esemény is egy időpontban kezdődik, csak más teremben, a felhasználó, ezen megoldás alkalmazásával, az események közül könnyűszerrel választhat, hogy melyiken szeretne résztvenni. Az összehasonlításra megoldást nyújt az események böngészésének lehetősége is.

A második megjelenítés már sajátosabb megoldás, hiszen egyetlen teremre leosztott események időrendi felosztását téríti vissza. Az utóbbi megoldás, abban az esetben hasznos, ha a fesztiválon résztvevők egy bizonyos teremben történő események iránt érdeklődnek.

Az események, mindkét megoldás esetén napok szerint szűrhetőek.

Minden *táncos* számára elérhető a rendezvény fő chatje, amelyen keresztül egyszerű üzeneteket küldhetnek egymásnak és értesülhetnek a rendezvény programját illető változásokról. Ezek mellett, privát táncfelkéréseket küldhetnek egymásnak. Egy beérkezett felkérésre válaszolhatnak. A sikeres felkéréseket, amennyiben mindkét fél befejezettnek jelenti, opcionálisan ki is értékelhetik. Ezek az értékelések publikus adatokként elérhetőek a *táncos* felhasználók által, amelyekből megismerhetik egymás képességi szintjét egy adott táncstílusban.



1. ábra. A *táncos* szerepkör korábbi (szürke) és újonnan (sárga) bevezetett funkcionalitásai.

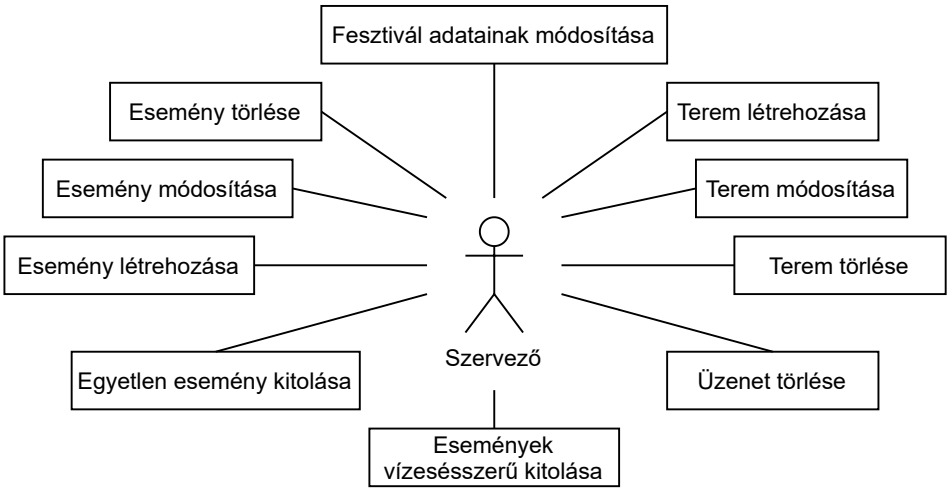
1.2. Szervező

A *szervező* szerepkör magában foglalja a *táncos*hoz rendelt összes funkcionalitást, emellett más feladatkörökkel is rendelkezik. Fő feladata menedzselni a fesztiválhoz tartozó adathalmazt. Ez egyaránt a fesztivál megtervezésekor és ennek lefolyásakor, valós időben történik. Jelen esetben a szervező csak módosíthatja a fesztiválhoz tartozó alap információkat, mint például a leírást, kezdeti és végidőpontot, a megjelenő táncstílusok listáját.

Emellett a *szervező* a rendezvény építőblokkjait módosíthatja, törlheti, illetve újakat is felvezethet. Ilyen entitások a termek és események. A termek esetén a fő táncstílus mellé másodlagos stílusokat is megjelölhet a felhasználó, illetve személyre szabhatja egy szobaszín definiálásával. Az események létrehozásakor a szervező meghatározza a helyszínt, az időpontot, a terjedelmét és az előadót. Egy esemény csak egy teremhez rendelhető és létrehozáskor vagy egyszerű módosításkor az eseményátfedés nem megengedett művelet.

Az események hosszabbítására, kétféle művelet áll rendelkezésre. Az első megoldás egy esemény periódusához hozzáad egy rövid időtartamot és amennyiben a kitolt esemény így konfliktusba lépne az őt követő eseménnyel, az utóbbi tartamát megrövidíti és kezdeti ideje egy későbbi időpontra helyezi. Ez az eljárás alkalmas, amennyiben a kifutó eseményt követő esemény nem igényli a megszabott időtartamot teljesen kihasználni.

A második változat nem csak a kitolt eseményt követő eseményre hat ki, hanem szükségszerűen az aznap megtartandó összes eseményre is. A kiválasztott esemény időtartamának megnövelését követően, minden őt követő eseménynek a kezdeti időpontját egy



2. ábra. A *szervező* szerepkör funkcionalitásai.

későbbi alkalomra teszi, tartamukat változatlanul hagyva. Ez komolyabb és előreláthatatlan problémákra, mint rossz időjárás, áramszünet vagy előadók késésére nyújt megoldást.

Mindkét esetben a *szervező* és a *táncos* felhasználók, a már korábban említett chatben kiküldött értesítések által ismertetve lesznek valós időben ezekről a változtatásokról.

2. Az alkalmazás felépítése

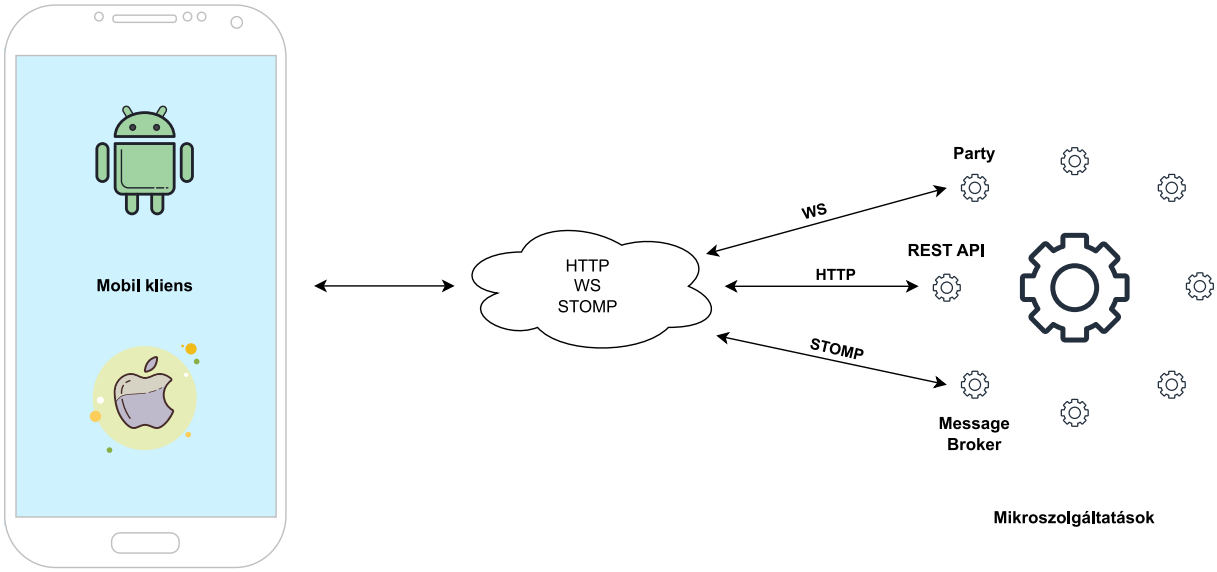
Jelen fejezetben bemutatásra kerül az alkalmazás architektúrája, a mikroszolgáltatásokon alapuló szerver és a cross-platform technológiát használó mobilalkalmazás, illetve a köztük megvalósuló kommunikáció.

2.1. Kommunikáció

A szerver és a mobilalkalmazás közti kommunikáció három protokollt követve épül fel, ezek a *HTTP* [17], *Websocket* [9] és *STOMP* [11] (lásd 3. ábra).

A felhasználók kezelése, hitelesítése, a fesztivállal kapcsolatos információk, a termek és az események részleteinek lekérése, valamint a táncfelkérések lebonyolítási folyamatának egy része *HTTP* hívásokon keresztül valósul meg. A kliens egy *REST* (Representational State Transfer) konvenciókat követő API-val [2] (Application Programming Interface) kommunikál, amely elengedhetetlen egy rugalmas, skálázható és könnyen megközelíthető architektúra kialakításához. Az adatok DTO-kon [7] (Data Transfer Object) keresztül áramlanak, amelyek hozzájárulnak az adatok helyességének fenntartásában és a rétegek egyértelmű elhatárolásában.

Az alkalmazásba beépített, valós idejű üzenőfal megvalósítására a *WebSocket* kommunikációs protokoll használt, amely valós idejű adatátvitelt tesz lehetővé a mobilalkalmazás és a szerver között egy állandó, kétirányú kapcsolaton keresztül. Ez a fajta kommunikáció, direkt módon, a Party szolgáltatás és a mobil kliens között zajlik.



3. ábra. Használt protokollok: HTTP, WebSocket, STOMP

A táncfelkéréssel kapcsolatos értesítések adatáramlását a *STOMP* protokoll valósítja meg. A STOMP egy üzenet alapú protokoll, amely szöveges üzenetek küldésére használt. A táncfelkérés lebonyolítása kezdetben egy REST hívással indul, amely a szerver belső felépítését követve egy *event* formájában eljut a táncfelkéréseket intéző mikroszolgáltatáshoz (Messaging Service), amely STOMP protokollt használva eljuttatja a táncfelkéréssel kapcsolatos üzenetet a címzetthez (lásd 2.2.2. fejezet).

2.2. Szerver

A *Wanna Dance?* szoftverrendszer magját egy mikroszolgáltatás architektúrára [13] épülő szerveralkalmazás képezi. Egy fesztivál applikáció estén számos előnnyel szolgál egy ilyen architektúra kialakítása: *skálázhatóság, ellenálló képesség, javított felhasználói élmény*.

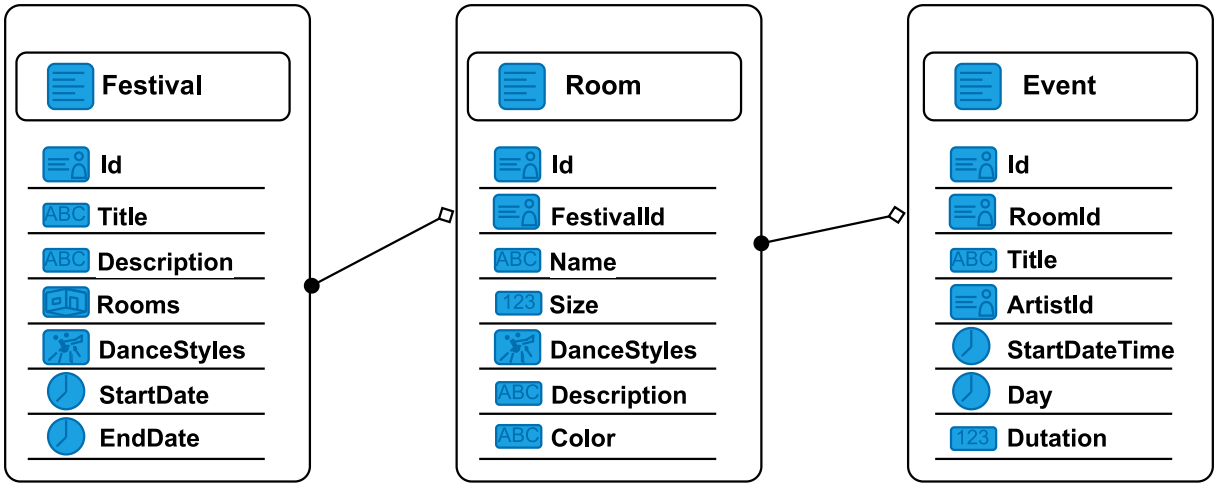
A mikroszolgáltatások egymástól függetlenül *skálázhatók*, lehetővé téve, a fesztivál alatt, a forgalom ingadozás könnyű kezelését. Például, ha magasabb forgalom várt napközben az esemény információk lekérdezésére, akkor skálázhatja az erre szolgáló mikroszolgáltatást anélkül, hogy az egész alkalmazást skáláznia kellene, így nem csak erőforrásokat spórol, hanem költséghatékonyabbá is válik.

Egy mikroszolgáltatás-architektúrában minden szolgáltatás független, ami azt jelenti, hogy ha egy szolgáltatás meghibásodik, nem fogja az egész alkalmazást leállítani. Ez javíthatja az alkalmazás *ellenálló képességét* a fesztivál alatt, ahol váratlan problémák adódhatnak. Mivel ezek jól meghatározott feladatot ellátó, apró programok, az újraindítás gyors, illetve párhuzamosan több példány futtatása zökkenőmentessé teheti az esetleges leállások kiküszöbölését.

Mivel minden mikroszolgáltatás jól elhatárolt, ezért ezek külön-külön *fejleszthetőek és telepíthetőek*. Ez gyorsíthatja a fejlesztési folyamatot, és lehetővé teszi a frissítések vagy javítások gördülékenyebb telepítését. Akár az alkalmazás teljes leállítása nélkül is be lehet iktatni ezeket a javításokat, amelyek élesben, egy fesztivál ideje alatt rendkívül hasznosak lehetnek.

Az alkalmazás felosztásával kisebb, könnyebben kezelhető szolgáltatások hozhatók létre, amelyek gyorsabb és reagáló-képesebb felhasználói felületet biztosíthatnak. A szolgáltatások a projektben, az entitások és funkcionalitások alapján vannak szétbontva. Ennek és a keretrendszer biztosította terhelés elosztásnak (Load Balancing) köszönhetően előfordulhat, hogy esetleges leállásokat nem is érzékelnek a felhasználók.

2.2.1. Adatmodellek



4. ábra. Entitások és a köztük lévő kapcsolatok

Annak érdekében, hogy az applikáció fesztiválok támogatására alkalmas legyen, környezeti elemzés után, a következő entitások lettek definiálva: fesztivál, terem és esemény (lásd 4. ábra). Jelenleg csak egy fesztivál szervezése lehetséges, viszont a szerveroldali implementáció több fesztivál menedzselését is támogatja.

A szerveralkalmazás felépítésére a *Moleculer* keretrendszer (lásd 3.1. fejezet) lett felhasználva. A keretrendszer biztosít adatbázis specifikus modult, amelyen keresztül elérhető az adatbázis, továbbá biztosítja az alapvető CRUD műveleteket, vagyis létrehozás (**C**reate), lekérdezés (**R**ead), frissítés (**U**pdate) és törlés (**D**elete). Ez a modul egy JavaScript mintát követő, úgynevezett *Mixin*be [14] burkolódik. Ez a minta lehetővé teszi az objektumok közötti funkciók újrahasznosítását anélkül, hogy azok közvetlenül az objektumok osztályába lennének beépítve. Így az objektumok sok különböző mixinből származó funkciót is felvehetnek, amelyeket az alapértelmezett funkciókon kívül használhatnak. Ezek alapján valósul meg a *Database per Service* [3] minta is. Mindegyik szolgáltatás egy entitás karbantartásáért felelős, ami fontos a jól izolált szolgáltatások létrehozásában, hiszen a gyakorlatban előfordulhat, hogy ezek a szolgáltatások különböző környezetekben futnak. Az entitások közötti kapcsolatok megvalósítása is a szolgáltatások szintjén történik, ami szintén az elhatárolódás célját szolgálja.

2.2.2. Mikroszolgáltatások

A korábban említett három új entitás (lásd a 2.2.1. fejezet) bevezetéséhez három új szolgáltatásra van szükség. Annak érdekében, hogy a rendszer valóban ellenálló legyen a

hibákkal szemben, ezek a szolgáltatások külön környezetben futnak. A szolgáltatások közötti kommunikáció megvalósítására a *transporter* modul felelős, amelyet a keretrendszer biztosít. A HTTP kéréseket egy "átjáró" szolgáltatás kezeli, ami az API Gateway mintát hivatott megvalósítani. Az API Gateway minta egy központi vezérlő pontot biztosít az alkalmazások közötti kommunikációhoz, amely elősegíti az egységesített hozzáférést a szolgáltatásokhoz. Ez a szolgáltatás az útvonal alapján eldönti, hogy melyik specifikus szolgáltatásnak kell továbbítania a kérést. A szerver felépítése az 5. ábrán tekinthető meg.

Végigkövetve tehát, a következő lépéseken halad át egy REST lekérdezés:

- A kérés megérkezik az átjáróhoz (Gateway) ami felméri, hogy jelen van-e lokálisan a keresett szolgáltatás;
- Ha jelen van, akkor a lokális kommunikációs buszt használva továbbítja a kérést egy megfelelő szolgáltatásnak, ellenkező esetben a transporter modulon keresztül;
- A megfelelő szolgáltatás feldolgozza a kérést és a válasz visszajut a klienshez.

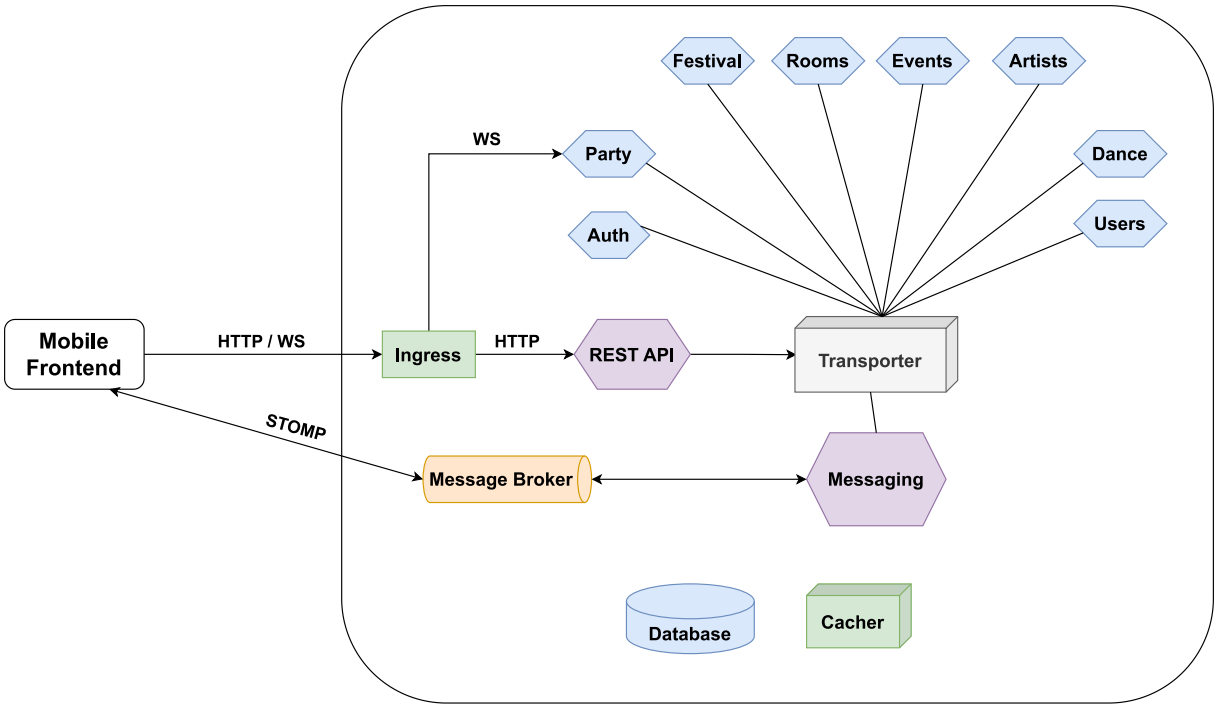
Ennek köszönhetően a kliensnek nem kell tudnia, hogy melyik szolgáltatással kommunikál, így teljesen független lehet tőlük.

A kétirányú kommunikációs kapcsolatok (WebSocket), direkt módon, a Party szolgáltatás és a mobil kliens között valósulnak meg. Ez a szolgáltatás az üzenőfal és az aktív felhasználók számontartásáért felelős. Az ilyen jellegű kommunikáció megvalósítására az *Ingress*, egy *Kubernetes* erőforrás (resource) használt, ugyanis maga a kitelepítés is ennek a rendszernek a segítségével valósul meg.

A táncfelkéréssel kapcsolatos értesítések lekezelésére egy újabb, különálló szolgáltatás használt (Messaging Service). Egy táncfelkérés lebonyolítása egy *HTTP* hívással indul, amit a Táncfelkérések szolgáltatás kezel le. Miután ez a kérés a keretrendszer nyújtotta *event* segítségével továbbítódik az üzenetküldő szolgáltatáshoz, az a RabbitMQ segítségével kiküldi a táncfelkéréssel kapcsolatos üzenetet (felkérés, elfogadás, elutasítás stb.) a címzettnek.

2.2.3. Biztonság

Ahogy már a szerepköröknél is említve volt (lásd 1. fejezet), különböző jogosultságú felhasználók használják a szoftverrendszert, ezért a szerver végpontjai védettek a megfelelő jogosúltsággal nem rendelkező kliens kérésekkel szemben. A kliensnek egy megfelelő JWT tokenre (JSON Web Token[8]) van szüksége minden kérés alkalmával, amit hitelesítés után



5. ábra. Szolgáltatások felépítése

kap meg. A JWT token három részből áll. Az első rész a Header, amely azt írja le, hogy az elektronikus aláírás elkészítésére milyen algoritmus használt. A második rész a Payload, ami tartalmazza JSON formátumban a konkrét adatokat, vagyis felhasználónév, hozzáférési jogosultság (szervező, táncos), azonosító és lejárási idő. A harmadik rész a Signature, amely tartalmazza az elektronikus aláírást. Azért, hogy URL paraméterként könnyen átadható legyen, a token három része *Base64url Encoding*[3] kódolással van kódolva. Ezek a tokenek minden kérés alkalmával dekódolásra kerülnek, hogy elkerüljük azt, hogy egyszerű felhasználók olyan funkcionalitásokhoz férjenek hozzá, amikre nem jogosultak.

Ezek az ellenőrzések köztes rétegekben, úgynevezett *Middleware*-ekben, történnek, amiket szintén natívan támogat a keretrendszer. Csak szervező szerepkörrel rendelkező felhasználó tud adatokat bevezetni a fesztiválla, termekkel, eseményekkel kapcsolatban. Ezeket más felhasználók csak olvasni tudják. Táncfelkérésekre viszont, bármilyen szerepkörű felhasználó képes, így akár a szervezők is intézhetnek egy-egy táncfelkérést az alkalmazáson belül.

[3] A Base64url Encoding - A Base64 kódolás azon formája, amikor az URL-ben használatos "+" és "/" jeleket kicseréljük rendre "-" és "_" karakterekre, valamint elhagyjuk a padding karaktert (az egyenlőségjelet, azaz "=").

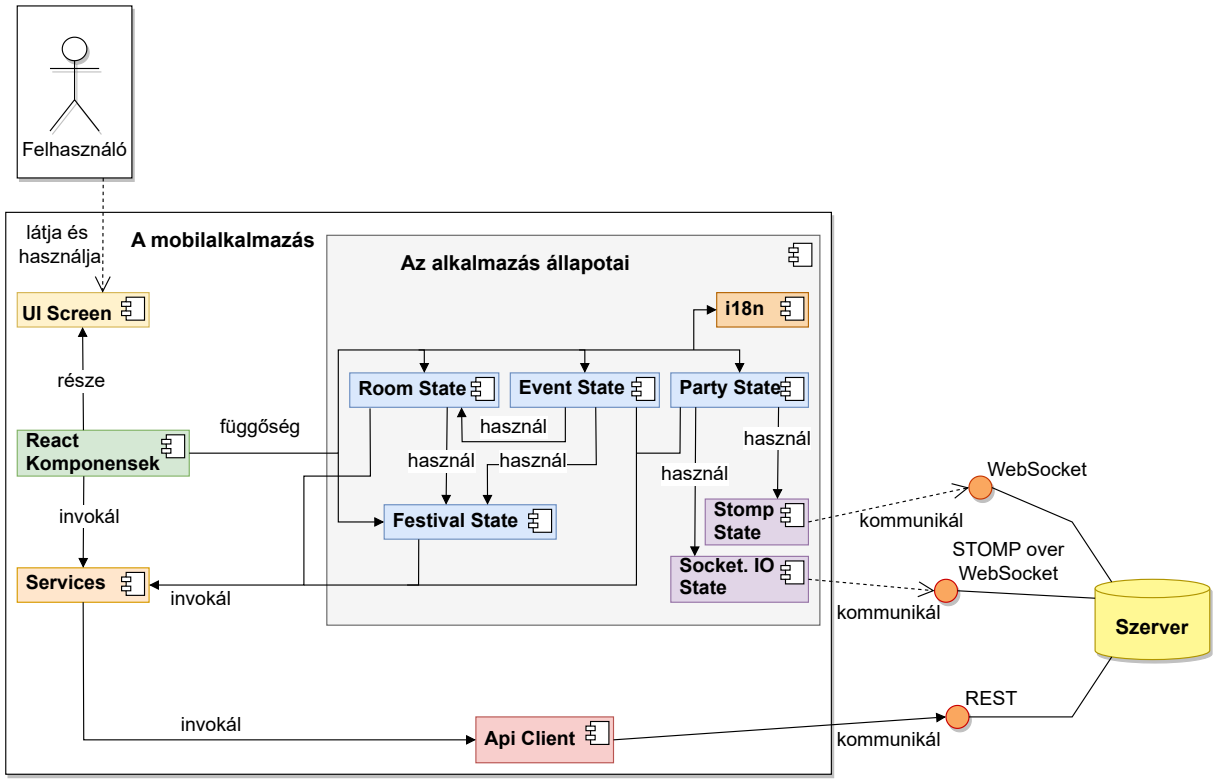
2.3. Mobilalkalmazás

A szoftverrendszer prezentációs rétegét képezi a mobilalkalmazás, amely a korábban bemutatott kommunikációs protokollokon keresztül kommunikál a szerverrel. Az alkalmazás *React-Native*-ben [10] íródott. A *React Native*, a Reacthoz hasonlóan, *JavaScript*ben [5] megírt építőblokkszerű komponensekből állítja össze az alkalmazást.

2.3.1. Adatpállapot menedzsment

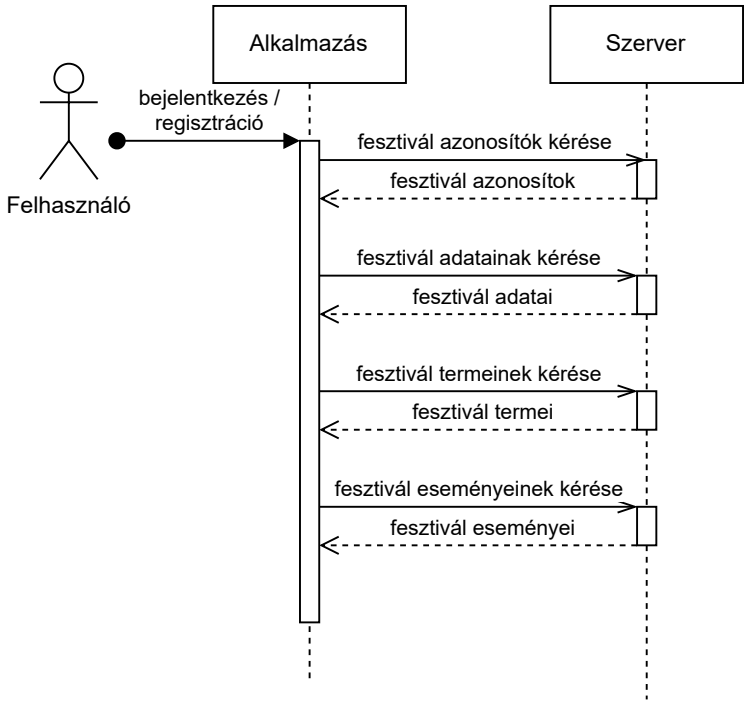
Az információk gyors megjelenítésének érdekében, a rendezvénnyel és felhasználókkal kapcsolatos adatok globális állapotokban vannak eltárolva az alkalmazáson belül. Ehhez az *unstated-next* [1] könyvtár van alkalmazva, amellyel különböző konténerek hozhatóak létre. Minden nagyobb entitásnak létezik egy neki megfelelő konténer a kód szintjén, amelyekben adatok mellett, adatmanipuláló metódusok is jelen vannak. Ezek a konténerek, a React Native által felhasznált DOM[4]-nak a legfelső szintjein helyezkednek el, annak érdekében, hogy a belső komponensek szintjén könnyen hozzáférhetőek és megoszthatóak legyenek az adatok.

Minden konténernek megfelelő komponens a *State* nevezetű modulban található. Ezek



6. ábra. A mobilalkalmazás architektúrája

[4]DOM - Document Object Model



7. ábra. Adatok betöltése bejelentkezés után

mellett még találhatóak olyan állapotok, amelyek az alkalmazás indulásakor a kommunikációs csatornákat állítják készenlétbe a *useEffect* által, amely a React egyik beépített hookja [4], ilyenek a *SocketIoState* és a *StompState* (lásd 6. ábra). Ezek a metódusok nagyrészben API[5] hívásokat végeznek a szerver felé. Az ilyen REST kérések, mint a GET, POST, PUT, PATCH és DELETE [16] egy külön *Service* rétegen találhatóak. Ezen metódusok használatakor a rendszer először szerveroldalon próbálja az adatmódosító műveleteket végrehajtani, majd sikeres válasz esetén, ezek a módosítások végbemennek a globális állapotokban eltárolt adatokon.

A rendezvénnyel kapcsolatos adatok szekvenciálisan kérődnek le a szervertől. Először a fesztivál azonosítója, majd annak általános információi töltődnek be a *FestivalState*ben. Ezt követően a fesztivál azonosítója alapján a termek a *RoomState*be és az események az *EvenState*be kerülnek betöltésre (lásd 7. ábra).

2.3.2. Biztonság

Lévén, hogy az alkalmazás nem csak a rendezvény eseményeinek követésére alkalmas, hanem azok valós idejű szervezésére is, számos képernyő és funkcionalitás elérhetetlen a alapfelhasználók számára.

Bejelentkezést követően, a felhasználó adatai, mint annak neve, felhasználóneve,

[5]API - Application Programming Interface

táncban betöltött szerepe és felhasználói szerepköre, lementődnek a globális szintű *LoggedInUserState*be. Ezeken kívül, egy *Hozzáférési Token*t (Access Token) is lement, amely az alkalmazáson keresztül kiküldött API kérésekhez van rendelve (lásd 2.2.3. fejezet). A szerver minden kérésnél végez egy token vizsgálatot, amely eredményének függvényében szolgálja ki azokat.

Különböző komponensek csak abban az esetben jelennek meg, ha a bejelentkezett felhasználó rendelkezik egy bizonyos jogosultsági szinttel (ebben az esetben, ha *szervező* az illető). Ezen komponensek legtöbbje navigációs szereppel rendelkeznek, amelyek által olyan képernyőkhöz lehet eljutni, ahol például fesztivállal kapcsolatos adatokat lehet létrehozni, törölni vagy módosítani, ami csak a *szervező*knek megengedett.

2.3.3. Routing

Az alkalmazáson belüli navigáció a *React Navigation* könyvtárral lett megvalósítva. Ennek alkalmazásához, az applikáció egy *NavigationContainer* objektummal van beburkolva. Ez lehetővé teszi, hogy képernyők közötti navigáláskor további adatok továbbítódjanak, függetlenül a komponensek felépítésétől.

Azon képernyők, amelyek valamilyen entitás részletesebb bemutatására vagy azok módosítására szolgálnak, valójában előre elkészített sablonokat használnak, melyek az átnavigálást követően megkapott azonosítók szerint töltik be a megfelelő adatokat.

A könyvtár kétféle képernyő típust biztosít: *TabScreen* és *StackScreen*. A *TabScreenek* közötti navigáció egy alsó TabBar használatával működik, amely jelen marad az összekötött *TabScreenek* közötti koordináláskor, míg a *StackScreenek*, a nevükből adódóan egymásra tevődve jelennek meg.

Az alkalmazás három navigációs csoportból épül fel: *Main Stack Navigation*, *Festival Stack Navigation* és *Party Tab Navigation*. A navigáció ezen csoportok képernyői között közrezáró és szomszédsági viszonyokon keresztül történnek.

2.3.4. Nemzetköziesítés

A mobilalkalmazás jelenleg három nyelvet támogat, angol, magyar és román, amelyek közül az angol az alapértelmezett. A nemzetköziesítés két könyvtár segítségével lett megvalósítva, a korábban megemlített *unstated-next*el és a *React Intl*-vel [12].

Az *unstated-next* által egy globális szintű állapotban van tárolva az aktív helyszínhez kötött nyelv. A globális állapotnak köszönhetően, nyelvváltoztatás esetén az alkalmazás minden érintett komponense befrissül és a kiválasztott nyelvnek megfelelő szövegeket jeleníti meg. Ezeket a megfelelő *JSON*[6] állományokból próbálja betölteni egy azonosító szerint, amennyiben ez sikertelen, az alapértelmezett nyelvhez járuló JSON-ból tölti be a hiányzó részeket.

[6]JSON - JavaScript Object Notation

3. Felhasznált eszközök és technológiák

Ez a fejezet bemutatja a projekt fejlesztése alatt használt fontosabb eszközöket és technológiákat.

3.1. Szerver technológiák

Keretrendszer

Ahogy már egy korábbi fejezetben említve volt (lásd 2.2.1. fejezet), a szerver megvalósítására a *Moleculer.js* nevű keretrendszer szolgált segítségül.

A *Moleculer.js* egy modern, gyors, mikroszolgáltatásokra épülő keretrendszer *Node.js*-hez. A fejlesztőknek hatékony, skálázható és hibatűrő osztott rendszerek készítését teszi lehetővé. A keretrendszer támogat többfajta kommunikációs protokollt és beépített funkciókat nyújt a terheléselosztásra, hibatűrésre és szolgáltatás-nyilvántartásra (lehetőség arra, hogy a szolgáltatások dinamikusan felfedezzék egymást). A Moleculer.js lehetővé teszi a fejlesztőknek, hogy egyszerűen és hatékonyan készítsenek mikroszolgáltatás alapú alkalmazásokat.

A szolgáltatások közötti kommunikációért felelős transporter a *NATS* rendszer segítségével valósul meg.

Adattárolás

Minden szolgáltatás külön gyűjteménnyel (collectionnel) rendelkezik, ami segít ezek elhatárolódásában. Erre a *MongoDB* van alkalmazva, ami egy gyors, dokumentum alapú, NoSQL adatbázis. Ebben az esetben előnyös volt egy egyszerű adatbázissal dolgozni, hiszen az entitások közötti kapcsolatok kialakítása amúgy is a szolgáltatások szintjén történik. A MongoDB továbbá könnyű skálázhatóságot és rugalmasságot nyújt, ami elengedhetetlen a mikroszolgáltatások szempontjából.

Tekintve, hogy egy fesztivál applikációról van szó, a gyors adatáramlás fontos szerepet kap. Ennek megvalósítása érdekében gyorsítótárazás alkalmazása ajánlott. Erre a *Redis*, egy nagy teljesítményű, kulcs-érték párokra alapuló adatbázis használt. A Redis egyedisége, hogy a memóriában tárolja az adatokat, ami lehetővé teszi a gyors hozzáférést és frissítést.

Kommunikáció

Az üzenőfalnál megvalósításához a *Socket.io* könyvtár használt, amely lehetővé teszi a valós idejű, kétirányú kommunikációt a kliens és a szerver között, valamint a táncfelkérésekkel kapcsolatos értesítéseket a *RabbitMQ*, egy üzenetküldő (message broker) szoftver biztosítja.

3.2. Mobil technológiák

Keretrendszer

A szoftverrendszer mobilalkalmazás része egy platform-független keretrendszeren alapszik, amely a *React Native*. A React Native olyan komponenseket és API-kat kínál, amelyek lehetővé teszik olyan JavaScript kód írását, amely átalakul natív platform-specifikus kóddá, vagyis ugyanaz a kódbázis használható a fejlesztéséhez mind iOS és Android eszközökön.

A React Native natív komponensek felhasználásával működik, amelyek az adott platform sajátosságait használják ki (View, Text, Image, stb). Ez lehetővé teszi az alkalmazás számára, hogy gyorsan és hatékonyan fusson, míg ugyanakkor biztosítja a natív "look and feel"-t és a magas felhasználói élményt.

A React-hez hasonlóan itt is komponensekből épül fel az alkalmazás, ami lehetőséget nyújt más, külső könyvtárak beemelésére. Ilyen például a *React Navigation*, amely az alkalmazáson belüli navigációért felelős, *React Native Elements*, amely alap komponensek egységes kinézetét biztosítja, vagy akár külső, közösségi könyvtárak, amelyek segítenek, hogy általános célú komponenseket ne kelljen újra lefejleszteni (dátum kiválasztó, toast típusú üzenetek, stb).

Adatállapot menedzsment

Amint már a 2.3.1. fejezetben említve volt, a komponensek közötti egyszerű adatátvitelhez globális állapotok használtak. Ennek megvalósítására az *unstated-next* könyvtár nyújt segítséget, amely kifejezetten React alkalmazásokban használt, mert *React Context*-en alapul.

Kommunikáció

A három kommunikációs protokoll megvalósítására dedikált könyvtárak használtak. A HTTP hívásokhoz *Axios*, a valós idejű adatátvitelhez a *Socket.io*, valamint a táncfelkérésekkel kapcsolatos értesítésekhez a *Stomp.js* könyvtár van alkalmazva.

Nemzetköziesítés

A nemzetköziesítéshez pedig a *React Intl* könyvtár használata nyújtott megoldást, amely lehetővé teszi a szövegek lokalizálását és a dátumok, idők és pénznemek formázását. A könyvtár a szövegek fordításához JSON formátumú fájlokat használ, amelyek a nyelvek és a szövegek megfelelő párosításait tartalmazzák.

3.3. Eszközök és módszerek

Kitelepítés

A szerveroldali technológiáknál említett négy modul (NATS, RabbitMQ, Mongo, Redis) *Docker* segítségével futnak. A Docker olyan virtuális konténereket hoz létre, amelyek elkülönítik az alkalmazások futtatását a gazda rendszer többi részétől. Az alkalmazásokat és a függőségeket tartalmazó konténer egy olyan izolált környezetet biztosít, amely lehetővé teszi az alkalmazások egyszerű telepítését, futtatását és karbantartását.

Ez elősegíti a kitelepítési folyamatot, amely *Kubernetes* segítségével zajlik. A Kubernetes lehetővé teszi osztott rendszerek rugalmas és hatékony üzemeltetését konténerekben. Ennek köszönhetően az alkalmazások és a függőségek könnyen telepíthetőek és karbantarthatók.

Agilis projektmenedzsment

Az agilis projektmenedzsment és szoftverfejlesztés egy rugalmas, iteratív fejlesztési folyamat, amely az együttműködést helyezi előtérbe. A résztvevők folyamatosan értékelik és módosítják a projekt irányát és a végtermék követelményeit. Az agilis módszertan alkalmazkodóképes és jól alkalmazható változó környezetben és követelményekben. Számos módszertan és keretrendszer létezik az agilis projektmenedzsment és szoftverfejlesztés területén, mint például a Scrum.

Scrum keretrendszer

A *Wanna Dance?* szoftverrendszer fejlesztése alatt a *Scrum* keretrendszer volt alkalmazva mint agilis munkamódszer.

A Scrum a fejlesztést úgynevezett sprintekre bontja, amelyek tipikusan 1-4 hetes időszakokra osztják fel a fejlesztési ciklust. A sprinteket a csapatok a végtermék meghatározott

részeinek fejlesztésére és tesztelésére használják.

A Scrum keretrendszer alapvető elemei a következők:

- A *Scrum Master* felelős a folyamat felügyeletéért és az agilis módszertan megfelelő alkalmazásáért.
- A *terméktulajdonos* felelős a végtermék követelményeinek meghatározásáért és az ügyfél igényeinek kielégítéséért.
- A *fejlesztői csapat* felelős a végtermék fejlesztéséért és teszteléséért a sprinteken belül.

A Scrum folyamat során a csapatok napi stand-up meetingeket tartanak, amelyek során áttekintik az előző napi munkát, az aktuális munkát, valamint az esetleges problémákat és kihívásokat. A sprintek végén a csapatok egy demo-t tartanak, amely során bemutatják a végtermék aktuális állapotát és az elért eredményeket, valamint reflektálnak a saját teljesítményükre.

A Scrum keretrendszer előnye, hogy lehetővé teszi a csapatok számára a hatékonyabb kommunikációt és együttműködést, valamint az ügyfél igényeinek pontosabb megértését és kielégítését. A Scrum segít a csapatoknak gyorsabban reagálni a változó környezetben és a követelményekben, valamint javítja a termékek minőségét és hatékonyságát.

A fentebb említett munkamódszer volt alkalmazva a fejlesztés alatt. A mentorok a Scrum Master és a terméktulajdonos szerepét öltötték magukra. A sprintek két hetes intervallumokban zajlottak, követve annak konvencióit: sprint tervezés, napi megbeszélések, demo és visszatekintés.

Verziókövetés, kódelemzés és tesztelés

A projekt kódbázisa *Gitlab* repositorykban vannak tárolva. A Gitlab egy DevSecOps[7] (Development, Security, Operations) platform, amely lehetővé teszi a fejlesztők számára, hogy verziókezeljék a forráskódjukat és problémamentesen együttműködjenek egy projektben.

A GitLab rendelkezik egy *CI/CD* szolgáltatással, amely folyamatos integrációt nyújt, automatizálva a munkamenetet, biztosítva a statikus kódelemzést a tesztelést és a kitelepítést. Ez segíthet időben kiküszöbölni az esetleges hibákat.

[7] DevSecOps - Egy olyan fejlesztési folyamat, amelyben a biztonság már a tervezési és fejlesztési szakaszokban is figyelembe van véve.

A Gitlab *pipeline*ok segítségével folyamatosan figyeli a forráskód változásait és automatikusan végrehajtja a meghatározott lépéseket, úgynevezett job-okat, a változások kezelésére. A *Wanna Dance?* alprojektek esetében a *lint* és a *test* job-ok aktívak, melyek statikus kódelemzők és unit tesztek futtatásában merülnek ki.

A statikus kódelemzők olyan szoftverfejlesztési eszközök, amelyek az alkalmazás forráskódjának elemzésével automatikusan észlelik a lehetséges hibákat, anomáliákat és a kód minőségével kapcsolatos problémákat, még az alkalmazás futtatása előtt. Kódminőség ellenőrzésre az *EsLint* szoftver használt, amely JavaScript kódbázisokhoz elemzésére készült.

Unit tesztelésre a *Jest* keretrendszer használt, amely az automata tesztelést segíti elő. Ezek mellett *End to End* (E2E) tesztek is használtak, melyek a *Detox* könyvtár segítségével valósulnak meg. A Detox kifejezetten React Native alkalmazások teszteléséhez készült. Az E2E tesztek az alkalmazás egészének működését, vagyis a mobil UI-t és a szervert egyaránt tesztelik, beleértve a hálózati kommunikációt és az adatbázis műveleteket is.

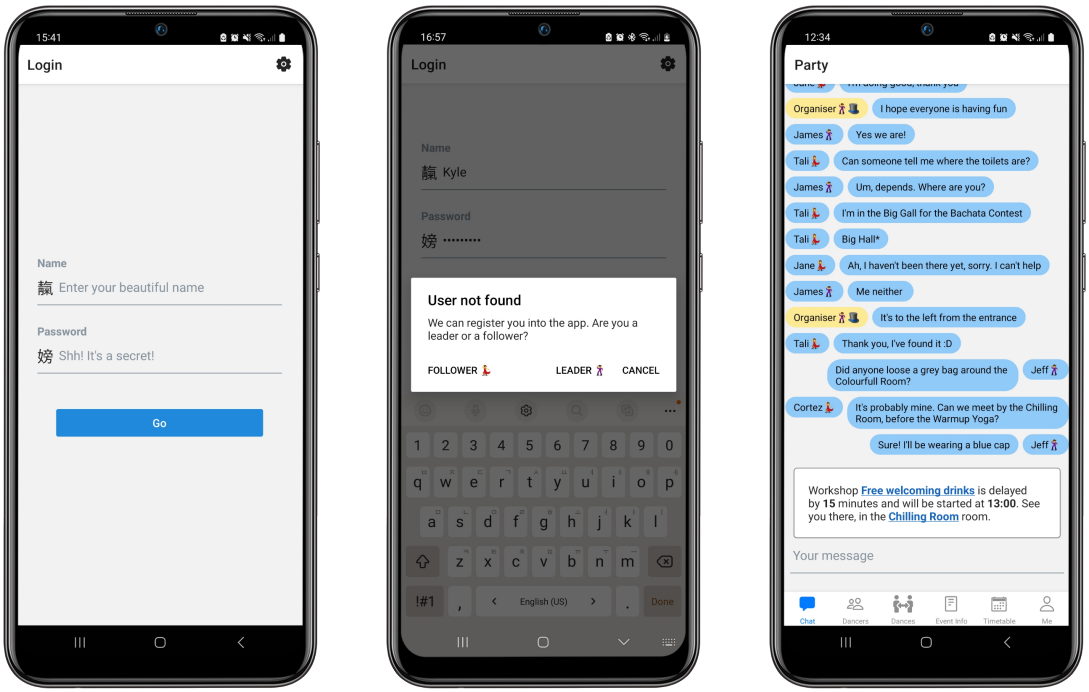
4. Az alkalmazás működése

Az alkalmazás megnyitásakor a felhasználót egy bejelentkezési oldal fogadja (lásd 8a. ábra). Ezen oldalon végezhető el a regisztráció is, amennyiben a megadott felhasználónév nem foglalt már egy létező aktív felhasználó által. Fiók létrehozásakor az új felhasználó kiválaszthatja a páros táncokban betöltött szerepét, amely lehet vezető vagy követő (lásd 8b. ábra).

Sikeres bejelentkezést követően a *Csevegő* képernyő (lásd 8c. ábra) jelenik meg és láthatóak lesznek a korábbi beszélgetések és értesítések. Mivel ez egy végtelen beszélgetőfal, az alkalmazás indításakor az utolsó ötven darab üzenet van lekérve, optimalizálva az alkalmazás működését. A lekérés után elküldött üzenetek hozzáadódnak a jelenlegi fal aljára. A felhasználó a szövegmezőre érintve üzenetet írhat, ezt elküldve, minden egyes felhasználó számára látható lesz.

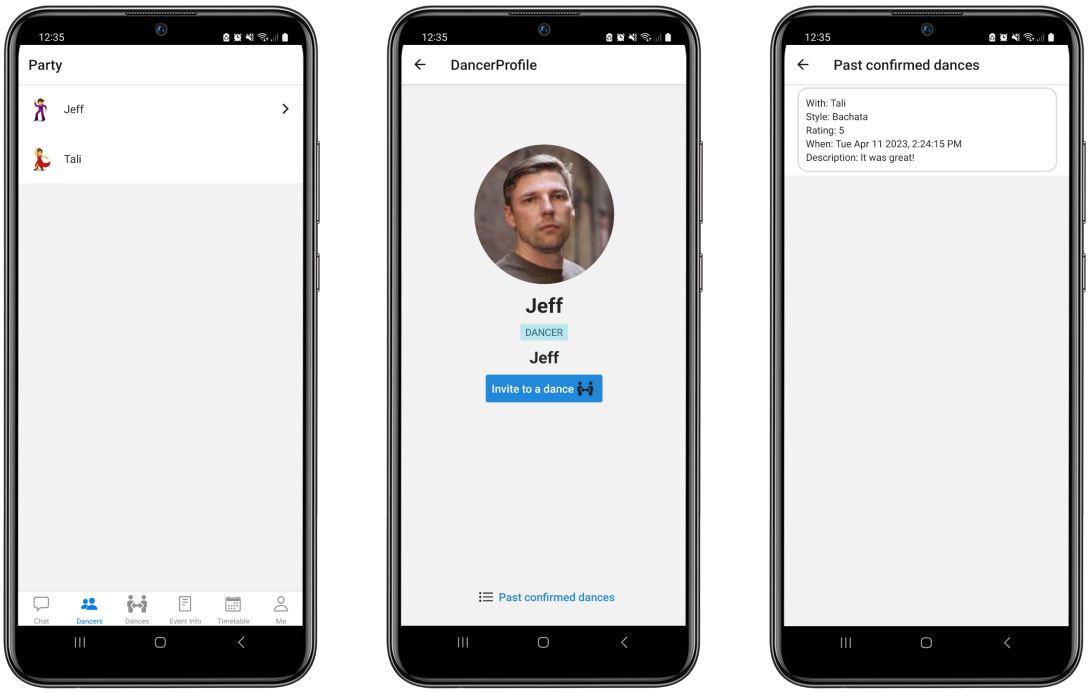
Egy kiválasztott üzenet hosszú érintésére egy lebegő ablak jelenik meg, amelyen, a bejelentkezett felhasználó jogosultságától függően, következő műveletek jelenek meg: az üzenet másolása, ez minden bejelentkezett felhasználó számára elérhető, és az üzenetek törlése, amely kizárólag csak *szervezők* által végrehajtható.

A képernyő alján található *navigációs sávval* a következő képernyőkre lehet navigálni: *Táncosok*, *Táncok*, *Rendezvény Információk*, *Órarend*, *Én*.



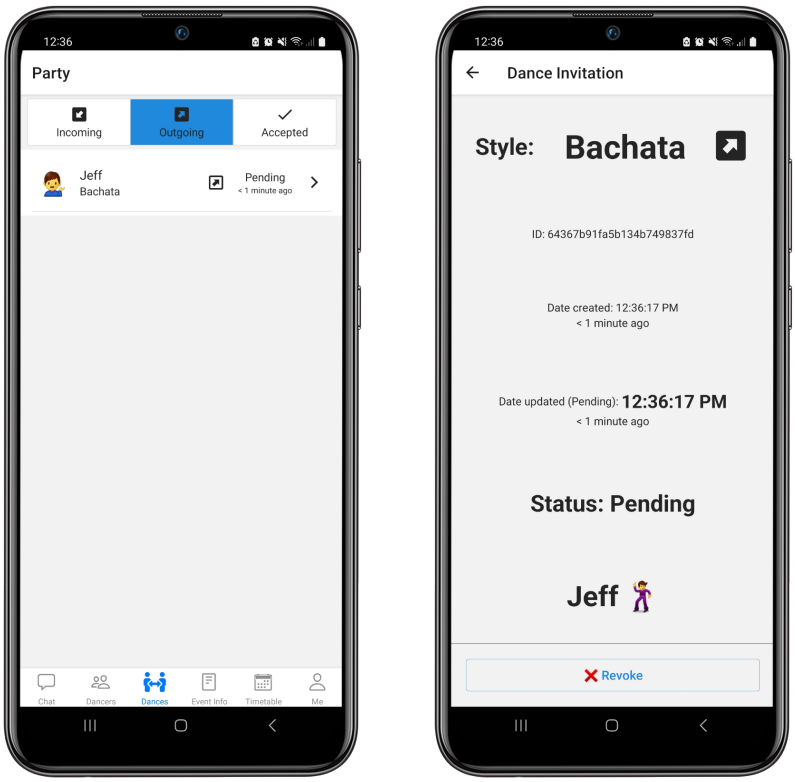
(a) (b) (c)

8. ábra. A *Bejelentkezés* (a), *Regisztráció* (b) és *Csevegő* (c) képernyők



(a) (b) (c)

9. ábra. A *Táncosok* képernyőn (a) keresztül elért *táncos profil* (b) és előző táncok listája (c)



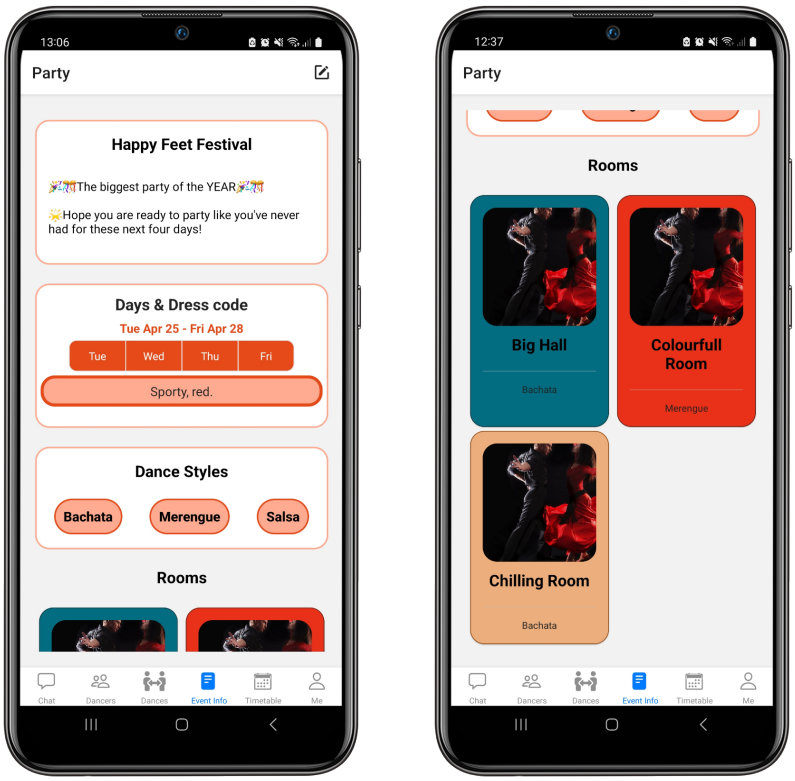
(a) (b)

10. ábra. Kiküldött táncfelkérések a *Táncok* képernyőn (a) és annak részletezése (b)

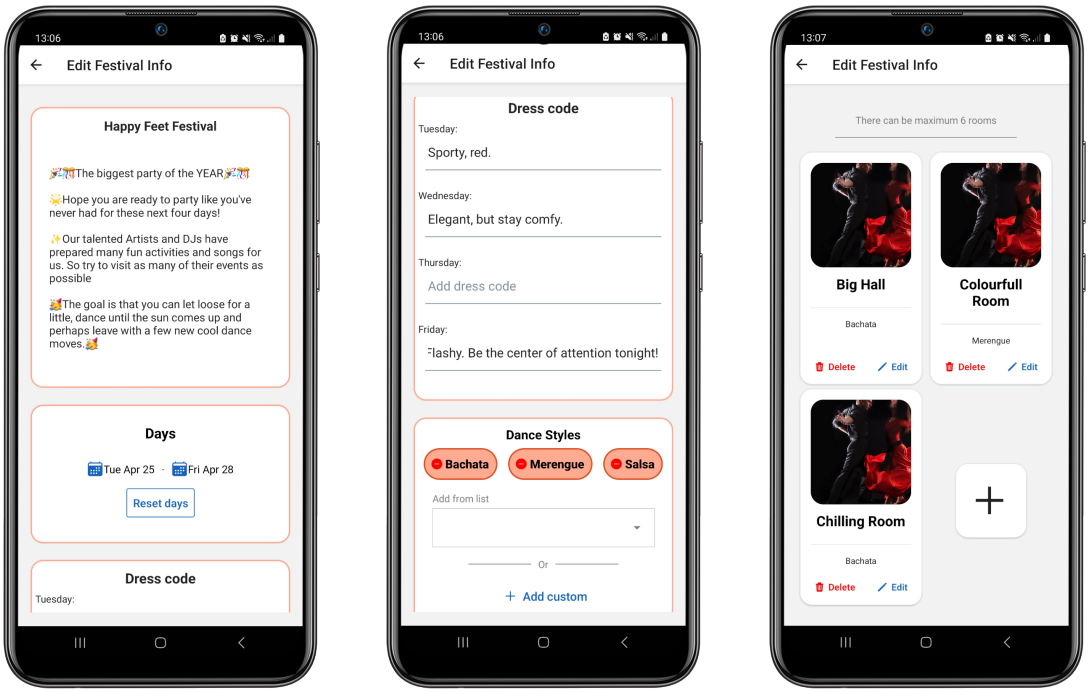
A *Táncosok* képernyőn a többi, fesztiválon résztvevő és bejelentkezett felhasználó listája található. A lista elemei révén, a felhasználó átnavigálhat egy másik felhasználó saját oldalára (lásd 9b. ábra), ahol megtekintheti eme személy előző táncain hagyott értékeléseket (lásd 9c. ábra). Az értékelésekből kiderül ezen felhasználó tudásszintje egy adott táncstílusban, illetve, hogy melyek a kedvelt stílusai. Abban az esetben, ha megfelelő társnak titulálja a felhasználó, akkor a profilon található *Meghívás Táncra* (Invite to a dance) gombra kattintva felkérést küldhet.

Egy táncos a hozzá kapcsolódó táncfelkéréseket a *Táncok* képernyőn tekintheti meg, ahol ezek három kategóriába vannak besorolva: Bejövő (Incoming), Kimenő (Outgoing) és Elfogadott (Accepted) (lásd 10a. ábra). Ezen táncfelkéréseket, az elfogadásukat követően, mindkét fél konfirmálhatja a tánc befejeztét és kiértékelhetik azt, ha szeretnék (lásd 10b. ábra). Abban az esetben ha mindkét táncos kiértékelést hagy a táncon, a visszajelzések publikussá válnak a további felhasználók számára is.

A fesztivál információi az *Rendezvény Információk* képernyőn találhatóak meg, ahol ezek típusuk szerint csoportosítva vannak. Elsősorban az általános információk vannak feltüntetve, mint a rendezvény neve, leírása, főbb táncstílusai, időtartama, valamint a napokhoz rendelet



11. ábra. *Rendezvény Információk* képernyő

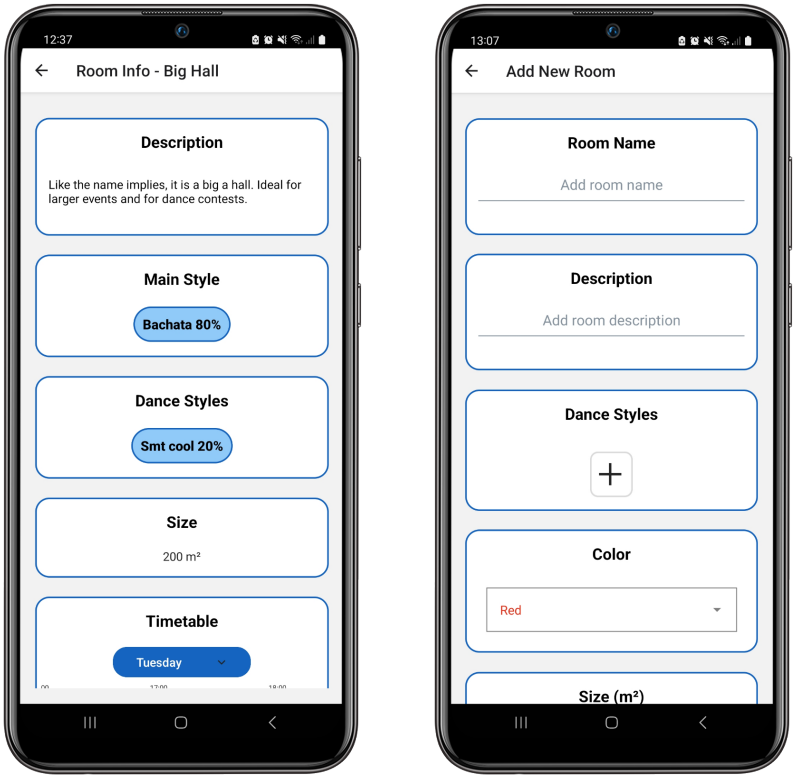


12. ábra. *Rendezvény Információkat Módosító* képernyő

opcionális viseleti szabályzat. Ezeket követik a rendezvény által használt termek (lásd 11. ábrák). Egy teremkártya megérintésével annak a részletező képernyője jelenik meg, olyan információkkal, mint a terem neve, leírása, mérete, a teremre vonatkozó táncstílusok és a teremhez rendelt szín (lásd 13a. ábra).

Mint *szervező*, az előbb említett két képernyőn, a *Rendezvény Információk* és az egy termet részletező *Terem Információk* képernyőn, egy kis ceruza ikonnal feltüntettet gomb által ezen információk módosító képernyői érhetőek el (lásd 12. és 13b. ábrák). Ezeken *input* mezők, naptárak és egyéni *droplist*ek segítségével vihetők végbe a műveletek. Valamint egy lebegő ablakon keresztül saját táncstílus is bevezethető. A rendezvény időtartamának módosításakor további adatok is módosulhatnak közvetett módon, amelyekre az alkalmazás felhívja a szervező figyelmét. Azon öltözködési szabályok és események, amelyek az új korlátokon kívül esnek, automatikusan törlésre kerülnek. Hasonló módon törlődnek egy teremhez tartozó események is, ha a terem törlődik.

Az entitások módosítása esetén, a mezők változtatásakor folyamatos validáció történik, mint például szöveghossz, fesztivál időtartam, név egyediség vagy akár események átfedésének az ellenőrzése. Abban az esetben, ha az új érték megfelel az elvárásoknak, akkor az érték, mező páros elküldődik a szervernek és végbemegy a részleges entitás módosítás.



(a) (b)

13. ábra. *Terem Információk* (a) és *Terem Létrehozó* (b) képernyő

A korábban említett termekhez tartozó eseményeket egyaránt meg lehet tekinteni a teremhez tartozó képernyőn vagy, az erre a célra kifejlesztett, *Órarend* képernyőn (lásd 14a. ábra). Egy teremhez tartozó események, azzal egy sorban vannak elhelyezve, egy görgethető panelen. Ez a panel egy időrácsnak felel meg, amelyen egy nap van felosztva tízperces intervallumokba, reggel tíztől egészen a következő napba átnyúló hét óráig. A termek feletti lenyitható elemből kiválasztható, hogy melyik nap eseményei legyenek megjelenítve. Az oldal felső jobb sarkában pedig egy újratöltési gomb található, amely a rendezvénnyel kapcsolatos összes információt lekéri, a 2.3.1 fejezetben felvezetett módon.

A felhasználók rendelkezésére áll még egy órarend stílus, amelyet egy teremkártya megérintése által érhetnek el. Itt csakis egy bizonyos szobára vonatkozó események vannak felrajzolva, vertikálisan. A napok szerinti szűrés itt is elérhető (lásd 14b. ábra).

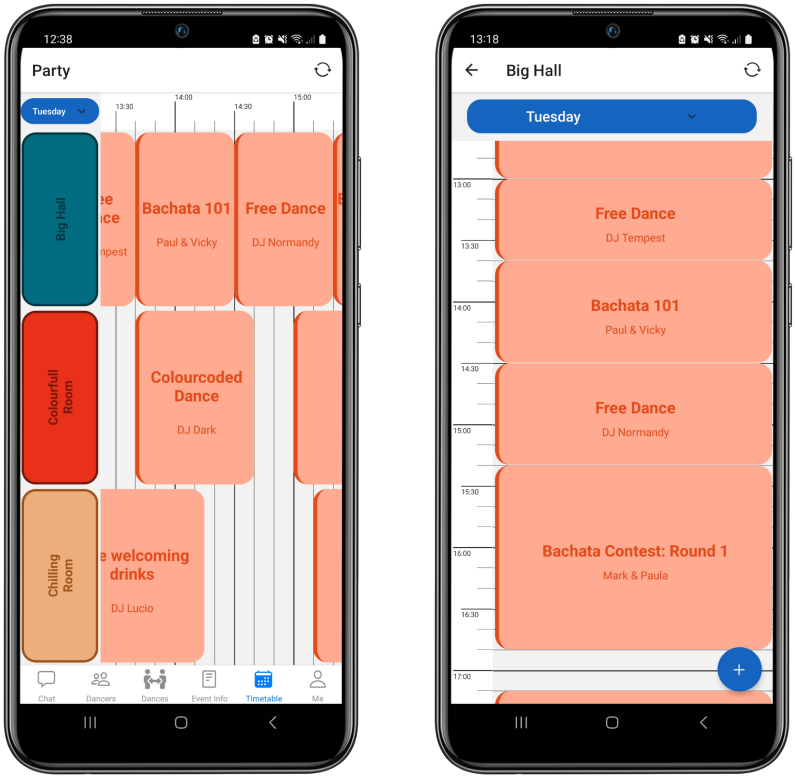
A termekhez hasonlóan, egy ilyen eseménykártya érintése egy részletező *Rendezvény Információk* képernyőre irányít. Az órarendről leolvasható információk találhatóak, egy kényelmesebb formátumban (lásd 15a. ábra).

Mindemellett, az előző képernyőkhöz hasonlóan, a szervezőknek lehetőségük van egy eseményt módosító képernyőre navigálni. Az esemény időtartamát befolyásoló műveletek

esetében, az alklamazás jelzi, ha egyes módosítások által átfedések jönnének létre, valamint egy, az órarendhez hasonló ábrán követhető, hogy ezek a módosítások, hogy néznek ki (lásd 15b. és 15c. ábrák). Maga egy esemény létrehozása is hasonlóan működik. Az ehhez szükséges képernyő pedig az *Órarend* képernyőn lévő lebegő gombbal érhető el, amely szintén jogosultság függvényében jelenik meg.

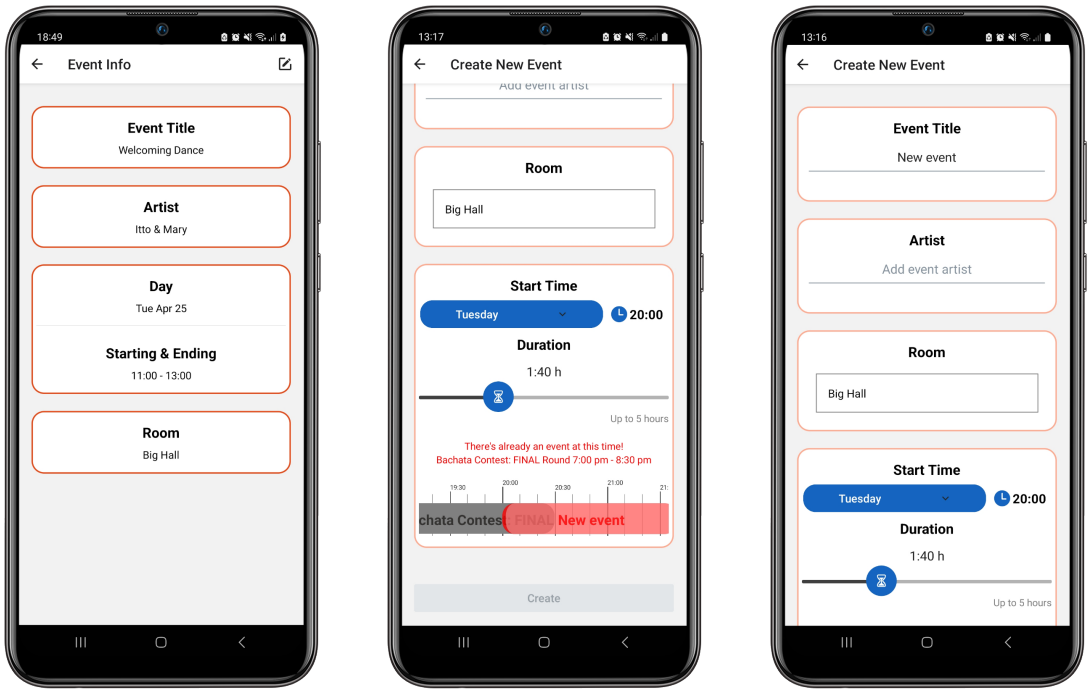
Szervezőként a felhasználó számára további lehetőségek is elérhetőek az *Órarend* nézeten. Egy eseménykártya hosszúérintésekor overlay jelenik meg, amelyen újabb két opció válik elérhetővé. Az egyik az esemény törlésére a szolgál, a másik pedig egy esemény kitolására (lásd 16a. ábra). A kitolás elvégzése egy lebegő panelen keresztül történik (lásd 16b. ábra). A felhasználó meghatározza a szükséges kitolási időtartamot a csúszka segítségével, majd eldöntheti az eltolás típusát a jelölőnégyzet kijelölésével. A művelet ellenőrízhető a panelen megjelenő ábrán.

Mindkét kitolás esetén a kiválasztott esemény időtartama bővülni fog a megadott időtartammal, a szervező feladata, hogy eldöntse, hogy milyen mértékben befolyásolja ez a művelet a rákövetkező eseményeket. Az alapértelmezett kitolás a kiválasztott esemény időtartamát módosítja , majd az őt követő eseményeket, amelyekkel így átfedésbe kerülne,



(a) (b)

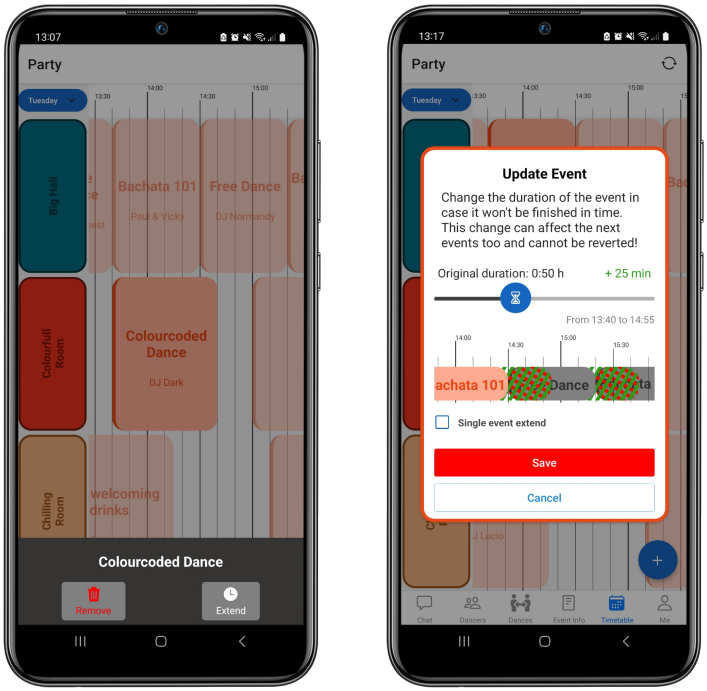
14. ábra. Horizontális (a) és Vertikális (b) Órarend képernyők



(a) (b) (c)

15. ábra. *Esemény Információk* (a) és *Esemény létrehozás* (b, c) képernyők

vízesésszerűen mozgatja el egy későbbi kezdeti időpontra, addig amíg a teremhez rendelet események között nem lesz több átfedés. Ebben az esetben minden utólagosan érintett esemény



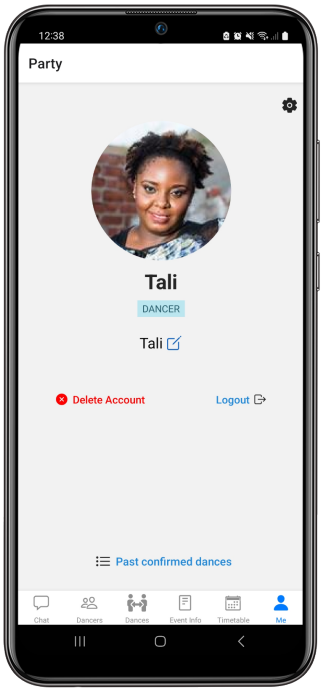
(a) (b)

16. ábra. Esemény overlay (a) és kitolás (b)

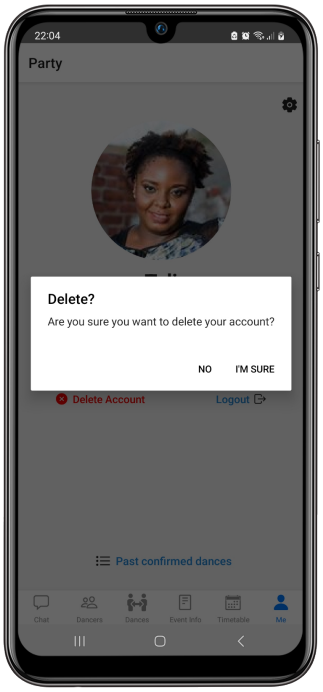
időtartama kötelezően változatlan marad. A második megoldás rövid időtartamváltozásra ajánlott. Az ilyen eltolás esetén, ha átfedés keletkezik, akkor a rákövetkező esemény kezdeti időpontját szintén el kell tolni. Itt a végidőpont változatlan marad, így az időtartam megrövidülését eredményezi ez a műveletsor.

Ezek az eseményeltolások valós időben, a fesztivál leforgása alatt történnek, ezért a rendszer figyelmeztető üzenetek formájában értesíti a felhasználókat a *Csevegő* képernyőn (lásd a 8c. ábra). Egy ilyen üzenet részletes leírást nyújt az érintett eseményről és az azon végzett módosításokról. Abban az esetben, ha egy módosítás több eseményt is érint, akkor egyenként minden eseményről érkezik egy éretsítés.

Nem utolsósorban, az *Én* képernyőn, egy felhasználó a saját profilját tekintheti meg (lásd 17. ábra), ez hasolnó a táncos profil nézethez (lásd 9b. ábra), viszont több lehetőség áll a felhasználó rendelkezésére. Itt módosíthatja a nevét a ceruza ikonra kattintva, ahol az előző módosító oldalakhoz hasonlóan, automatikusan kiküldődik az új név a szervernek. Ez a változás a Csevegő ablakban látható új üzenetek küldésekor. Emellett kijelentkezhet vagy végső esetben törölheti a fiókját (lásd 18. ábra). Saját táncait és a hozzájuk rendelt értékeléseket is megtekintheti a kijelölt nézeten. A felső sarokban levő fogaskerék által átléphet a *Beállítások* képernyőre, ahol kiválaszthatja az alkalmazás nyelvét, valamint a szerver elérhetőségi URL-jét módosíthatja (lásd 19. ábra). Ezutóbbi nem áll valós felhasználók rendelkezésére.



17. ábra. *Én* képernyő



18. ábra. Fiók törlése

19. ábra. *Beállítások*

Következtetések és továbbfejlesztési lehetőségek

A *Wanna Dance?* projekt főbb célkitűzései részben megvalósultak, hiszen a meglévő alkalmazásba sikerült bevezetni a fesztivál szervezést és nyomonkövetést.

Már az előzőlegesen megvalósított táncfelkérések lebonyolítása mellett, a rendszer lehetőséget nyújt a szervező számára felvezetni a fesztivállal kapcsolatos általános információkat. Emellett termek létrehozására, módosítására, valamint események bevezetésére és ezek késleltetésére is lehetőség van, amikről a táncosok az üzenőfalon keresztül értesülhetnek.

Bár jelenleg csak egy fesztivál szervezése támogatott, a projekt célja ezt kiterjeszteni több fesztivál menedzselésére párhuzamosan. A felhasználók több táncfesztivál között böngészhetnek és végigkövethetik azt, amelyiken éppen részt vesznek.

Ezek mellett egy olyan felület is tervben van a DJk[8] számára, amelyen keresztül valós idejű visszajelzést kaphatnak a táncosoktól a zenével kapcsolatban. A visszajelzésnyújtáshoz csak a mobilalkalmazás kiterjesztésére lenne szükség, viszont egy DJ számára egy webes felület lenne alkalmas. A két rendszer összekötésével a táncosok élőben reagálhatnak a zenére, vagy akár szavazhatnak is a következőkkel kapcsolatban.

Ahogy már említve volt, a táncosok az üzenőfal segítségével valós időben értesülhetnek az események tolódásával kapcsolatban. Ez viszont csak az alkalmazás aktív használatakor elérhető, visszatekinthető. Ezt kiterjesztve, alkalmazáson kívüli értesítések (push notification) által a felhasználók valós időben figyelmeztetve lesznek a programváltozásokról, így könnyebben tudnak reagálni ezekre az eseményekre.

Jelenleg a fesztivál, a termek és az események bemutatására csak szöveg alapú leírások szolgálnak. Ezeket képekkel lehetne kiegészíteni, ez által egy közvetlenebb betekintést nyújtva a táncosoknak, amely akár kedv fokozó hatással is bírhat.

Annak érdekében, hogy csak azok a felhasználók érhessék el az üzenőfal és táncfelkérés funkciónalitásokat, akik valóban a fesztivál területén tartózkodnak, különféle ellenőrzéseket kellene bevezetni: QR kód leolvasás vagy helymeghatározás.

A dolgozat részletezte a *Wanna Dance?* applikáció szerepköreit és funkciónalitásait (1. fejezet), bemutatta annak felépítését (2. fejezet), betekintést nyújtott a felhasznált eszközökre és technológiákra (3. fejezet), valamint kitéret a mobilalkalmazás működésére (4. fejezet).

[8]DJ (Disc Jockey) - egy olyan személy, aki zenét játszik különböző eseményeken, például klubokban, fesztiválokon

Hivatkozások

[1] Abdulazeez Abdulazeez Adeshina. *State management with Unstated Next*. 2020. URL: https://blog.logrocket.com/state-management-with-unstated-next/ (elérés dátuma: 2023. ápr. 19.)

[2] Leonard Richardson & Mike Amundsen. *RESTful Web APIs*. O'Reilly Media, 2013.

[3] Pietro Storniolo Antonio Messina Riccardo Rizzo. *The Database-is-the-Service Pattern for Microservice Architectures*. Springer, Cham, 2016.

[4] Daniel Bugl. *Learn React Hooks: Build and refactor modern React. js applications using Hooks*. Packt Publishing Ltd, 2019, 28–31. oldal.

[5] Győző Horváth és László Menyhárt. „Teaching introductory programming with JavaScript in higher education". *Proceedings of the 9th International Conference on Applied Informatics*. 1. kötet. 2014, 339–350. oldal.

[6] Melinda Tóth István Bege. „FestivApp: Program manager and browser system for large events". *IEEE SISY* (2016).

[7] Less Jackson. „Data Transfer Objects. In: The Complete ASP.NET Core 3 API Tutorial". *Apress, Berkeley, CA* (2016).

[8] Michael B. Jones, John Bradley és Nat Sakimura. *JSON Web Token (JWT)*. RFC 7519. 2015. máj. DOI: 10.17487/RFC7519. URL: https://www.rfc-editor.org/info/rfc7519.

[9] Andrew Lombardi. *WebSocket*. O'Reilly Media, 2015.

[10] Eric Masiello és Jacob Friedmann. *Mastering React Native*. Packt Publishing Ltd, 2017, 45–46. oldal.

[11] Jeff Mesnil. *Mobile and Web Messaging*. O'Reilly Media, 2014.

[12] Ibadehin Mojeed. *React Intl: Internationalize your React apps*. 2021. URL: https://blog.logrocket.com/react-intl-internationalize-your-react-apps/ (elérés dátuma: 2023. ápr. 19.)

[13] Irakli Nadereishvili. *Microservice Architecture*. O'Reilly Media, 2016.

[14] Addy Osman. *Learning JavaScript Design Patterns*. O'Reilly Media, 2012, 83–87. oldal.

[15] Farkasa Renáta. *A tánc lelkünkre gyakorolt hatásai*. 2021. URL: https://www.womagic.hu/lifestyle/a-tanc-lelkunkre-gyakorolt-hatasai/ (elérés dátuma: 2023. ápr. 19.)

[16] Leonard Richardson, Mike Amundsen és Sam Ruby. *RESTful Web APIs: Services for a Changing World*. " O'Reilly Media, Inc.", 2013, 33–39. oldal.

[17] Clinton Wong. *HTTP Pocket Reference*. O'Reilly & Associates, 2000.