ETDK-dolgozat

Szopos Dávid

Petkes Kinga

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

Kolozsvár, 2022. május 19–22.

SIRECO

Intézményeken átívelő foglalási rendszer

Szerzők:

Szopos Dávid

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

Petkes Kinga

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

Témavezetők:

dr. Simon Károly, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Balogh Szabolcs, szoftverfejlesztő,

Codespring

Gagyi Mátyás, szoftverfejlesztő,

Codespring

Vad Bertalan, szoftverfejlesztő,

Codespring

Kivonat

A SIRECO (Sharing Infrastructure and Resources for Events and Communities) projekt egy foglalási rendszert biztosít, amely intézmények infrastruktúrájának bérlését teszi lehetővé. Általában az intézmények rendelkeznek olyan belső rendszerrel, amely lehetővé teszi saját eszközeiknek és termeiknek foglalását, viszont a SIRECO alkalmazás egy több intézményt összefogó felületet biztosít, a publikusan is bérelhető termek és eszközök foglalásához.

A elsődleges célközönséget konferenciák, képzések, fesztiválok szervezői képezik, akik olyan eseményt szeretnének rendezni, aminek keretén belül párhuzamosan több épületben és több teremben zajlanak a programok. A másik célközönséget intézmények és bérelhető infrastruktúrát biztosító cégek képezik, akik a saját termeiket, és a termekhez tartozó eszközöket tehetik elérhetővé.

A szoftverrendszer rendelkezik egy központi szerverrel, valamint három kliensalkalmazással: egy iOS és egy Android alkamazással, melyek elsősorban a teremtulajdonosoknak biztosítanak gyors hozzáférést a foglalási kérésekhez, valamint egy webes felülettel, amely elsősorban a rendezvényszervezők számára nyújt hatékony keresési és foglalási lehetőségeket.

A dolgozat részletezi a projekt felépítését, illetve a fontosabb komponensek megvalósítását, kitérve a fejlesztés során felhasznált technológiákra és eszközökre, valamint bemutatja az alkalmazás működését.

Tartalomjegyzék

Bevezető **1**

1. Szerepkörök és funkcionalitások **3**

1.1. Nem regisztrált felhasználók 3

1.2. Regisztrált felhasználó 4

1.3. Intézménytulajdonos 5

1.4. Adminisztrátor 6

2. Architektúra **7**

2.1. Szerver 7

2.1.1. Adatmodell 7

2.1.2. Adathozzáférési réteg 9

2.1.3. API 9

2.1.4. Biztonság 11

2.2. Web 12

2.2.1. Kommunikáció a szerverrel 13

2.2.2. Komponensek 14

2.2.3. Routing 15

2.3. Mobil 16

2.3.1. Kommunikáció a szerverrel 16

2.3.2. Komponensek 17

2.3.3. Adattárolás 19

3. Felhasznált technológiák és eszközök **20**

3.1. Szerveroldali technológiák 20

3.2. Web technológiák 21

3.3. Mobil technológiák 21

3.4. Eszközök és módszerek 22

4. A SIRECO alkalmazás működése **25**

4.1. A webes felület használata 25

4.2. A mobil alkalmazás használata 29

Következtetések és továbbfejlesztési lehetőségek **33**

Bevezető

A nagyobb konferenciák, képzések és fesztiválok esetében, a párhuzamosan zajló programok lebonyolításához a szervezőcsapatoknak több, különböző méretű és típusú teremre, illetve megfelelő infrastruktúrára (vetítő, ledfal, hang- és fénytechnika, kamerarendszer stb.) lehet szüksége. Ezeket legtöbb esetben nem képes egyetlen intézmény biztosítani, a beszerzésük emiatt időigényes és komplikált feladattá válhat.

Az adott feltételek mellett a szervezők több platform átböngészésére kényszerülnek a megfelelő helyiségek megtalálása érdekében. Emellett, gyakran elkerülhetetlen a különböző intézmények tulajdonosainak telefonos felkeresése a pontos ár, a bérlés feltételeinek és a helyiségek felszereltségének megbeszélése miatt. Így mind a kiválasztás folyamata, mind a visszaigazolások követése körülményes, akár több embert is igénybevevő munka.

A fent említett bérelhető termekkel, illetve infrastruktúrával rendelkező intézményeknek, cégeknek általában létezik egy belső rendszere a foglalások kezelésére, de ehhez külső személyek nem férhetnek hozzá. Ugyanakkor sok esetben nem rendelkeznek megfelelő erőforásokkal egy különálló platform létrehozására sem, mely nyilvánosan is elérhetővé tenné a foglalás lehetőségét.

A SIRECO projekt célja megoldást nyújtani a fent említett problémákra, olyan felületeket biztosítva, melyek egyszerre könnyítik meg az események szervezőinek és a számukra termeket, segédeszközöket biztosító intézményeknek a dolgát. Az alkalmazás segítségével több intézmény ajánlatai tekinthetőek meg és foglalhatóak egy helyen a regisztrált felhasználók által, majd ezek a foglalások könnyen módosíthatóak, akár vissza is vonhatóak. Az egyes intézmények tulajdonosai bevezethetik bérelhető termeiket, megjelölve a rendelkezésre álló felszereléseket is, majd az ezekre érkező foglalási kéréseket egy gombnyomással igazolhatják vissza. Ezen kívül, egy adminisztrátor felel a spam tartalmak kiszűréséért, megakadályozva, hogy nem létező intézmények, vagy ajánlatok kerüljenek be a rendszerbe.

A szoftverrendszer három részből áll össze: egy webalkalmazásból, mely elsősorban a rendezvényszervezőknek nyújt segítséget a tervezési folyamatban, egy mobilalkalmazásból (iOS és Android kompatibilitással egyaránt), mely az intézménytulajdonosok dolgát könnyíti, és egy szerverből, mely a két kliensalkalmazás kéréseit szolgálja ki.

A dolgozat hátralevő része a következőképpen van struktúrálva: az 1. fejezet bemutatja az alkalmazásban elérhető funkcionalitásokat szerepkörökre lebontva, majd a 2. fejezet szemlélteti a szoftverrendszer architektúráját, külön tárgyalva a szerver, web, illetve mobil részeket.

A 3. fejezetben esik szó az egyes részek megvalósításához felhasznált technológiákról és eszközökről, a 4. fejezet pedig az alkalmazás működését mutatja be.

A projekt fejlesztése a U-Hub Mentorprogram keretein belül kezdődött el 2021 nyarán, a Codespring koordinálásával. A munkát folytattuk a 2021-2022-es tanévben a Csoportos projekt egyetemi tantárgy keretein belül, aminek ideje alatt csatlakozott a fejlesztői csapathoz Balázs Norbert, Kurunczi Viktória, Tesnádi Attila és Salló Attila. Ugyanakkor segítségünkre voltak mentoraink: Balogh Szabolcs, Gagyi Mátyás és Vad Bertalan, valamint témavezetőnk Dr. Simon Károly.

Ezúton szeretnénk megköszönni évfolyamtársaink hozzájárulását, mentoraink és témavezetőnk segítségét, szakmai irányítását.

1. Szerepkörök és funkcionalitások

A SIRECO projekt egy olyan egységes platformot biztosít, ahol a különböző intézmények könnyen elérhetővé tudják tenni a bérelhető eszközeiket és termeiket a megfelelő specifikációkkal. Ezáltal a platform megkönnyíti a nagyobb, több szekcióból álló és különböző helyszíneken zajló rendezvények lebonyolítását is a szervezők számára.

A rendszer két felhasználói felületet biztosít: a weboldal és a mobilalkalmazás. A funkcionalitások nagyvonalakban megegyeznek a felületeken, viszont a felhasználási területek különbözhetnek.

A mobilalkalmazás segítségével az intézménytulajdonosok értesítést kapnak a saját termeikhez érkező foglalásokról, így akár azonnal visszaigazolhatják őket. Emellett a fotófeltöltést vagy módosítást is egyszerűbbé teszi az applikáció, mivel helyben készíthetnek képeket az adott teremről. Továbbá, az adminisztrátoroknak is segítséget nyújt az alkalmazás, ugyanis hozzájuk érkeznek be az új intézmények hozzáadására vonatkozó kérelmek, és ezekre azonnal reagálhatnak.

A webes felület használatakor a rendezvényszervezőknek lehetősége van egy nagyobb program, esemény szervezésekor részletesebben böngészni, szűrni a termek között, ahhoz, hogy a számukra megfelelőket megtalálják. Ezt a műveletet lényegesen egyszerűbb egy weboldalon végezni, mint telefonon. Ráadásul, a saját foglalások esetleges módosítása, törlése vagy áttekintése is gördülékenyebb webes felületen, kevesebb hibalehetőséget hagyva.

A felhasználókat elsősorban négy fő kategóriába lehet elkülöníteni, mégpedig: *vendég* (nem regisztrált) felhasználó, *bejelentkezett* felhasználó, *intézménytulajdonos* és *adminisztrátor*. A fejezet részletesen tárgyalja a felhasználók jogosultságait, szerepkörökre lebontva.

1.1. Nem regisztrált felhasználók

Az alkalmazás használatához nem szükséges bejelentkezni, viszont ilyenkor csak bizonyos funkcionalitások lesznek elérhetőek, amint azt az 1. ábra is mutatja. Megtekinthető a termek listája a fontosabb információkkal, lehetőség van szűrni, keresni közöttük több szempont alapján, ezáltal a nem regisztrált felhasználó eldöntheti, hogy megéri számára regisztrálni vagy sem. Továbbá, ha bármi más funkcionalitást szeretne elérni, regisztrációra vagy bejelentkezésre lesz szüksége, amelyet megtehet az adatai megadásával, vagy a Google fiókjával.

1. ábra. Vendég felhasználó funkcionalitásai

1.2. Regisztrált felhasználó

Regisztrált felhasználók számára az alkalmazásban elérhető funkcionalitások listája kibővül a vendég lehetőségeihez képest. Ezt a 2. ábra szemlélteti.

Bejelentkezés után a meghirdetett termek részletes adatai is megtekinthetővé válnak, mint a pontos cím, a rendelkezésre álló felszerelések, egy rövid leírás a helyiség adottságairól, illetve egy képgaléria is. A megfelelő terem kiválasztásának folyamatát megkönnyítendő, naptár jeleníti meg a lefoglalt és szabad dátumokat, különböző színekkel jelölve a teljesen vagy csak részben foglalt napokat.

Amint az ideális tulajdonságokkal rendelkező és megfelelő időpontban bérelhető terem kiválasztásra került, a felhasználó elhelyezhet egy foglalási kérelmet, megadva az időintervallumot, mely alatt a helyiségre igényt tart. Az elküldött kérelmek és ezek állapotai - lehetnek várakozásban, elfogadva vagy elutasítva - a későbbiekben nyomon követhetőek. A felhasználóknak ugyanakkor lehetősége nyílik módosítani a már létező foglalásaikat, bármilyen állapotban is legyenek azok, megadva az új kezdeti, illetve befejezési dátumokat. Abban az esetben, ha valamilyen okból kifolyólag egy fenntartott teremre mégsincs szükség, a foglaló könnyen vissza is vonhatja foglalását.

Ezen felül, regisztrált felhasználók kérhetik intézményeik regisztrálását az alkalmazásba, amennyiben az rendelkezik nagyobb konferenciák, események lebonyolítására alkalmas, bérelhető termekkel.

2. ábra. Bejelentkezett felhasználó funkcionalitásai

1.3. Intézménytulajdonos

Az intézménytulajdonos rendelkezik az eddig bemutatott jogosultságokkal, ugyanúgy képes foglalni, böngészni termeket, továbbá joga van a saját intézményének kezelésére is. Egy adott szervezetnek, intézménynek több tulajdonosa is lehet, akik egyaránt képesek az intézmény termeinek a menedzselésére, karbantartására. Miután az adminisztrátor elfogadta a intézménykérelmet, a felhasználó, aki felvezette, intézménytulajdonossá válik. Ettől kezdve különböző termeket tud felvezetni az intézmény hatáskörén belül, amelyek kezeléséről gondoskodnia kell.

3. ábra. Intézménytulajdonos funkcionalitásai

Mint ahogy a 3. ábra is mutatja, a saját termek szerkesztése/módosítása ennek a szerepkörnek a feladata. A legfontosabb feladat a beérkező foglalások visszaigazolása, egy-egy foglalás elfogadása/elutasítása az intézmény lehetőségeinek és a foglalók igényeinek függvényében.

1.4. Adminisztrátor

Az adminisztrátor a vendég szerepkörét bővíti ki, ennek funkcionalitásait a 4. ábrán láthatóakkal egészíti ki.

Ahogy az fennebb említésre került, feladata a felhasználók által benyújtott intézmény-hozzáadási kérelmek visszaigazolása. Az adminisztrátor felelőssége tehát a felvezetett információk alapján elbírálni egy intézmény kérelmének indokoltságát, megelőzve ezzel a kéretlen tartalmak megjelenését. Egy intézmény visszautasításával a kérelmet elhelyező felhasználó szerepköre változatlan marad, elfogadás esetén viszont intézménytulajdonossá válik.

4. ábra. Adminisztrátor funkcionalitásai

2. Architektúra

A SIRECO szoftverrendszer három fő komponensből áll össze: egy React-ban [30] megírt webalkalmazásból, egy Expo [7] segítségével megvalósított mobilalkalmazásból, mely mind iOS-en, mind Android-on működőképes, és egy központi Node.js [24] API szerverből, mely RESTful [36] szolgáltatásként üzemel. Emellett a rendszer része az adatokat tároló MongoDB [21] adatbázis és a Firebase által biztosított Cloud Storage [13], mely a termekhez és felszerelésekhez feltöltött képek mentéséért felelős.

A fő komponenseket többrétegű architektúra [38] jellemzi, ezáltal a kód egyszerűbbé és átláthatóbbá válik. A megjelenő problémák kiküszöböléséhez és új funkcionalitások hozzáadásához nem szükséges az egész kódbázis újraírása, elég a megfelelő réteget módosítani, illetve az egyes rétegekben használt technológiák a későbbiekben könnyen lecserélhetőek az igényeknek megfelelően. Emellett ez a felépítés nagy mértékben megkönnyíti a fejlesztés folyamatát, mivel a csapat tagjai párhuzamosan dolgozhatnak a különböző rétegeken.

2.1. Szerver

2.1.1. Adatmodell

A szoftverrendszer adatait egy dokumentum alapú *NoSql* adatbázis tárolja. A relációs adatbázisok tábláinak soraihoz képest az adatok itt dokumentumokban tárolódnak, melyeket kollekciók fognak össze. A központi szerver *models* csomagjában vannak az entitások által követett sémák definiálva. Itt megjelennek a kollekciók által használt mezők, ezek típusai, alapértelmezett értékei és a betartandó megszorítások.

A SIRECO projekt öt különböző kollekciót definiál, ahogy azt az 5. ábra is mutatja. Ezek felépítése és felhasználása a következő:

- Az *Institutes* gyűjtemény tartalmazza az intézmények információit, mint a név, a borítókép elérési útvonala és az intézmény létrehozási kérelmének státusza. Az állapot értéke kezdetben *folyamatban lévő*, majd az adminisztrátor döntésének megfelelően veheti fel az *elfogadott* vagy *elutasított* értékeket.

- A *Users* kollekció tárolja a rendszer felhasználóinak adatait. Itt kötelezően megjelenik a kereszt- és vezetéknév, a felhasználónév, a hashelt jelszó, illetve a szerepkör, melynek értéke *foglaló*, *tulajdonos* vagy *adminisztrátor* lehet. Amennyiben egy felhasználó

5. ábra. A szerveroldalon definiált kollekciók és az egymáshoz viszonyított kapcsolatuk

intézménytulajdonos, a séma kiegészül az általa karbantartott intézmény azonosítójával. Ugyanakkor, a Google fiókkal belépő felhasználók esetében a külső szolgáltatótól kapott ID is mentésre kerül.

- A *Rooms* kollekció az intézmények által meghirdetett termek adatait tárolja. Megjelenik a termek neve, típusa, óránkénti bérlési ára, kapacitása, helyszíne (országot, várost és pontos címet is specifikálva), rövid leírása, borítóképe, a tulajdonos intézmény azonosítója és a rendelkezésre álló felszerelések listája. A felszerelések különböző értékeket tartalmazhatnak, például *vetítőt*, *hangosítási rendszert* vagy *ledfalat*.

- Az *Images* gyűjtemény a termekhez feltöltött képek Firebase Cloud Storage-beli elérési útvonalát tartalmazza a megfelelő terem azonosítójával együtt.

- A *Reservations* kollekcióban találhatóak a beérkezett foglalások. Ezek rendelkeznek kezdeti és befejezési dátummal, illetve státusszal. Az intézmény-regisztrálási kérelmekhez hasonlóan ennek is *függőben lévő* a kezdeti értéke, majd a terem tulajdonosának visszaigazolása után válik *elfogadottá* vagy *elutasítottá*. A séma emellett tárolja a terem azonosítóját, melyhez a foglalás tartozik, illetve a felhasználót, aki létrehozta.

2.1.2. Adathozzáférési réteg

Az adathozzáférési rétegben történik az adatbázissal való kapcsolattartás, az adatbázisműveletek végrehajtása, a lekérdezések küldése és fogadása. A SIRECO szerver MongoDB adatbázist használ, és ezzel a kommunikációt Mongoose [22] csomag segítségével valósítja meg. A Mongoose megkönnyíti az alkalmazás adatainak modellezését, és az adatbázis gyűjteményeinek, dokumentumainak megfelelően olyan objektumokat biztosít, amelyekkel műveleteket tudunk végezni a szerveren. Mongoose modellek létrehozásával az intézmények, foglalások, felhasználók adatai feldolgozhatóvá, kezelhetővé válnak. A modellekben az adattagok rendelkeznek különböző típusokkal, alapértelmezett értékekkel. Használhatunk *enum* típusú attribútumokat, így egy adott mező értékeit korlátozhatjuk a felsorolásban megadott értékekre, vagy *required* attribútumokat, kötelezővé téve egyes adattagokat.

Az adatbázishoz való hozzáférés egyszerű parancsokon keresztül történik, nincs szükség MongoDB szkriptek írására. Az adatmanipulációt a négy alapművelet, a *Create*, *Read*, *Update*, *Delete*, röviden CRUD műveleteken keresztül végeztük. Ez a négy alapművelet tulajdonképpen egy konvenciórendszer a tárolt adatok kezelésére, mely az állandó (perzisztens) adattárolást segíti elő, megkönnyítve az információk elérését/keresését.

A különböző entitások lekérésének optimális kezelését, és a memóriahasználat hirtelen növekedésének elkerülését a Mongoose *paginate* szolgáltatása oldja meg, ezáltal mindig csak bizonyos számú adat töltödik be, és jelenítődik meg a kliensnek, nem pedig az egész adathalmaz.

2.1.3. API

A projekt web- és mobilalkamazásait egy RESTful szolgáltatásként üzemelő API szerver szolgálja ki. A kommunikáció ennek értelmében HTTP protokollon alapszik, JSON formátumú üzenetekkel.

A beérkező kérések fogadása az Express Router middleware [9] feladata, a megírt Router-ek a **src/routes** csomagban csoportosulnak. Ezek feladata meghívni a megfelelő kontroller metódusokat, melyeket az **src/controllers** csomag tartalmazza.

A termekkel kapcsolatos kérések az **/api/rooms** elérési útvonalra érkeznek. Ezen keresztül listázhatóak a meghirdetett termek és hozzáadhatóak újak. A **/api/rooms/:id**-n keresztül lekérhető egy konkrét terem, ez módosítható, illetve törölhető. Ugyanakkor, az előbbi útvonal **/reservations**-el való kiegészítése által lekérhetőek az adott terem foglalásai. Az **/api/images**

elérési útvonal egy új kép hozzáadását teszi lehetővé, a JSON objektumban megadva a kép URL-jét és a megfelelő terem azonosítóját.

Az **/api/users** útvonal felelős a rendszer felhasználóinak listázásáért. A **/:id** változó megadásával elérhető a konkrét felhasználó, módosíthatóak az adatai vagy törölhető is. Az útvonal kiegészítése a **/rooms** kulcsszóval lehetőséget ad a felhasználó intézményéhez tartozó összes terem listázására, a **/reservations** kulcsszó pedig az adott felhasználó foglalásait teszi elérhetővé.

A bevezetett intézmények az **/api/institutes** útvonalon keresztül elérhetőek és regisztrálhatóak. Az **:id** megadásával tekinthetőek meg egy konkrét intézmény adatai, ezek módosíthatóak és eltávolíthatóak. Egy adott intézmény termeinek listázása az **/api/institutes/:id/rooms** útvonalra intézett kérés által lehetséges. Az **/api/institutes/:id/accept** és **/api/institutes/:id/decline** útvonalak pedig a függőben lévő intézmény-regisztrálási kérelmek visszaigazolásáért felelősek.

A foglalások az **/api/reservations** útvonalon érhetőek el és hozhatóak létre. Az **/:id** megadásával ebben az esetben is egy konkrét foglalás adatai kérhetőek le, módosíthatóak vagy törölhetőek a rendszerből. Emellett, a **/api/reservations/:id/accept** és **/api/reservations/:id/decline** útvonalak nyújtanak lehetőséget az intézménytulajdonos számára egy foglalás visszaigazolására.

Az **/auth** elérési útvonal felelős az autentifikációval kapcsolatos kérések kezeléséért. A **/register** elérési pont egy új felhasználó létrehozására ad lehetőséget, mellyel a **/login** útvonalon keresztül léphet be a felhasználó. A **/google/web/redirect** és **/google/mobil/redirect** pontokra pedig a Google fiókkal való sikeres bejelentkezéskor irányít át a külső szolgáltató. A **/logout**-ra érkező kéréssel a kijelentkezés valósul meg. Ugyanakkor, lekérhetőek a bejelentkezett felhasználó adatai a **/loggedInUserInfo** útvonalra intézett hívással.

A szerver és a kliensek közötti adatcsere a DTO tervezési mintát [4] követi. A minta alkalmazása több előnnyel is jár: csökken a szoftverrendszer adatforgalma, mivel az egyes DTO-k csak részleges reprezentációi a definiált modelleknek, illetve több adat gyűjthető össze egy válaszba, ezzel csökkentve a szerverhez érkező hívások mennyiségét. A felhasznált DTO-k és modellek közötti átalakítást megvalósító függvények a **src/util/dtos** csomagban találhatóak, modellek szerint kategorizálva.

2.1.4. Biztonság

A SIRECO alkalmazás használatához a felhasználónak nincs szüksége bejelentkezésre, viszont hogyha a böngészésen kivül bármi más funkcionalitást szeretne elérni, akkor be kell jelentkeznie a rendszerbe. Az 1. fejezetben leírt szerepkörökhöz különböző jogosultságok tartoznak, ezért ellenőrizni kell, hogy az adott kérést küldő felhasználó melyik szerepkörhöz tartozik, és hogy egyáltalán be van-e jelentkezve vagy sem.

A webalkalmazás esetében kétféleképpen lehet bejelentkezni: a felhasználó Google fiókjával, vagy egy regisztrációs űrlap kitöltése által.

A regisztrációs űrlap kitöltésekor minden szükséges információt kötelező megadnia a felhasználónak, mivel ezek által lesz azonosítva a későbbiekben. Az elküldött információk a szerver által hitelesítve lesznek, és csak akkor kerülnek elfogadásra, ha megfelelnek a követelményeknek. Elsősorban, a megadott felhasználónévnek egyedinek kell lennie. Továbbá, a jelszó megadását, az esetleges hibák kizárásának érdekében, két helyen várja el az alkalmazás, ha ezek nem egyeznek meg sikertelenné válik a regisztráció. Ha a megadott információk megfelelnek az elvárásoknak, a felhasználó adatai bekerülnek a 2. fejezetben említett adatbázisba. Sikeres regisztrációt követően, a bejelentkezéshez a felhasználónév-jelszó páros megadására van szükség, ekkor a szerver ellenőrzi, hogy a felhasználó megtalálható-e az adatbázisban. Amennyiben megtalálható, generál egy JSON Web Token-t (JWT) [19] a felhasználónak, amely tartalmazza a bejelentkező azonosítóját és felhasználónevét, szerepkörét, a hozzá tartozó intézmény azonosítóját, és a tokennek a lejárati idejét, titkosított formában.

A JWT egy nyílt szabvány, amely egy kompakt és önálló módszert határoz meg JSON objektumok segítségével az információk továbbítására a kommunikáló felek között. Használata biztonságos, mivel a benne rejlő információ digitális aláírással van ellátva, amelyet csak a feladó szerver ismer. A kompaktsága a kis méretére vonatkozik, mely lehetővé teszi, hogy könnyen továbbítható legyen URL-en keresztül, HTTP üzenet POST paramétereként, vagy a HTTP fejlécben.

Az API szerver ezt a tokent fogja visszaküldeni a kliensnek. A kliens ezáltal sikeresen be lesz jelentkezve, és ezután a kéréseinek a *header*-jében elküldésre kerül a generált JWT token is, amelynek segítségével a szerver azonosítani tudja, hogy az aktuálisan bejelentkezett felhasználónak milyen jogosultságai vannak. A token lejárati idejét követően újbóli bejelentkezésre van szükség.

Emellett, a mobil- és webalkalmazás esetében is lehetőségünk van Google fiókkal

6. ábra. Google fiókkal történő bejelentkezés

való bejelentkezésre, amely az *OAuth2.0* [25] protokollt használja a hitelesítéshez és az engedélyezéshez (lásd 6. ábra [26]). Első lépésként az alkalmazás átirányít a böngészőbe, vagy a telepített Google alkalmazáshoz, és a felhasználó bejelentkezését kéri a fiókjával. Ezek után a kliens egy kérést küld a Google felé, a Google pedig visszaküld egy bizonyos időre érvényes hozzáférési kulcsot, amelynek segítségével a kliens alkalmazás lekéri a felhasználóról az igényelt adatokat (thirdPartyID, név, felhasználónév). Ezek után a kliens elküldi az API szerver felé a megszerzett információkat, amely leellenőrzi a hozzáférési kulcs hitelességét, a kapott információkat, és ha először jelentkezik be a Google fiókjával, akkor az adatait elmenti az adatbázisba, különben csak frissíti. Ha sikeres a hitelesítés, akkor a felhasználó bejelentkezett, és annak függvényében, hogy milyen szerepkörrel rendelkezik, hozzáférhet az alkamazáshoz.

2.2. Web

A webalkalmazás feladata az API szerverrel való kommunikáció megvalósítása és a felhasználói felület megjelenítése. A web kliens a React [30] keretrendszerre épül, ez részletesebben a 3.2. fejezetben lesz bemutatva, és három fő csomagből tevődik össze, amelyek folyamatosan kommunikálnak egymással: *components*, *service*, *util*.

A **components** csomag tartalmazza a felhasználói felületen megjelenő összes komponenst. Ezek a komponensek alcsomagokba vannak struktúrálva, ezáltal sokkal áttekinthetőbb a kód, és az esetleges hibák könnyebben javíthatóak.

A **service** csomagban található függvények felelősek a szerverrel való kommunikációért, továbbá a szervertől kapott választ is ezek dolgozzák fel, és adják vissza a komponenseknek, a megfelelő formában.

A **util** csomagban levő szolgáltatások elősegítik az alkalmazás működését, konstansokat, segédfüggvényeket tartalmaznak, amelyeket a különböző komponensek használnak fel.

2.2.1. Kommunikáció a szerverrel

A webalkalmazás REST [36] kéréseken keresztül kommunikál a központi szerverrel, és a szerverhez való hozzáférés Fetch API [11] segítségével valósul meg. A Fetch API *promise*-kat használ a HTTP/1.1 protokollon keresztüli aszinkron kérések küldésére, az ezekre érkezett válaszokat pedig a **service** csomagban levő függvények kezelik. Minden entitáshoz külön tartozik egy megfelelő service fájl (reservationService, userService), ami kimondottan az adott entitáshoz tartozó műveleteket tartalmazza.

Ahogy a 1. kódrészleten is látszik, a Fetch kérésnek több paraméter is megadható, viszont ebből csak az elérési út (URL) kötelező. Emellett lehetőségünk van megadni a kérés *típusát* (valamilyen HTTP metódus), ami alapértelmezetten GET, *header* és *body* paramétereket, és mivel alaphelyzetben nem lesz továbbküldve a szerver fele a bejelentkezett felhasználót azonosító *token*, a *credentials* kulcsértékkel végrehajtja ezt a Fetch hívás. Tehát, ha olyan útvonalra szeretnénk kérést küldeni, ami védett adatokkal dolgozik, vagy bejelentkezést igényel, szükséges a *credentials* paraméter megadása. A bejelentkezett felhasználó adatai **context**-ben [3] lesznek tárolva, amely lehetőséget biztosít adatok átadására a komponensek között, anélkül, hogy minden szinten tovább kellene adni ezeket az információkat *props*-ként.

A Fetch kérés válaszul egy Promise-t fog visszaadni, amelynek a feldolgozását az *async/await* szintaxis könnyíti meg. Az 1. kódrészletben definiált aszinkron függvény valósítja meg egy intézmény hozzáadását: a kérés el lesz küldve a szerver fele, a beszúrandó intézménnyel a body-ban. Ha nem megfelelő a visszakapott válasz státuszkódja, akkor hibaüzenet fog megjelenni a felhasználó számára, különben vissza lesz térítve a szervertől küldött válasz, JSON objektum formájában, amelyet a *response.json()* [37] metódus dolgoz fel.

2.2.2. Komponensek

A SIRECO webalkalmazás összetett felhasználói felülete egyszerűbb részekre van bontva, amelyek megfelelnek az alkalmazás egy-egy felületének vagy oldalának, ezeket a részeket nevezzük komponenseknek. A *Components* csomag alcsomagokra van tagolva az átláthatóság és egyszerű módosítások érdekében, minden alcsomag csak a neki megfelelő komponenseket tartalmazza (*Room*, *Filter*, *Institutes*, *Reservation*, *Login*, *Map*, *RoomImages*). Például a *Room* alcsomag tartalmazza a termekkel kapcsolatos komponenseket, az *Institutes* az intézményekkel kapcsolatosakat, a *Filter* a különböző keresési, és szűrési komponenseket stb.

Ezek a komponensek betartják az *egyetlen felelősség elvét* (single-responsibility principle) [40], egy komponensnek csak egyetlen jól meghatározott szerepe van.

Az alkalmazás komponensei React függvénykomponensek [41], mivel JavaScript függvényeknek felelnek meg. Korábban másféle komponenstípusok használata is jellemző volt, de a React Hook-ok [31] bevezetésével lehetségessé vált, hogy a teljes alkalmazás függvénykomponensek segítségével legyen megírva. Ezek a függvények JSX-t [18] térítenek vissza, amely lehetőséget nyújt a JavaScript kód HTML szerű elemekkel való kombinálására. Ez megfigyelhető a 2. kódrészleten is, amely a *Components/Room/RoomDetails* komponens egy darabja, ahol egy teremnek a részletei lesznek megjelenítve, különböző kisebb komponensek segítségével. Például a *RoomDetailCard* komponens csak a konkrét terem adatait illusztrálja,

```
async function addInstitute(institute) {
  const response = await fetch(`${ENV.SERVER_URL}/api/institutes`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(institute),
    credentials: "include",
  });

  if (response.status === 500) {
    throw new Error(ALERTS.INSERTION_ERROR.FAILURE_MESSAGE);
  }

  return response.json();
}
```

1. kódrészlet. Példa a kliens által elküldött kérésre Fetch API segítségével, és a szerver által visszaküldött válasz feldolgozására

vagy a *MapContainer* komponens egy OpenStreetMap [27] alapú térképet jelenít meg, a terem lokációjával. Ugyanebben a példában látható, ahogy a JavaScript kód társítva van HTML elemekkel, mivel a térkép alapú megjelenítés csak abban az esetben történik meg, ha a terem lokációja meg volt adva a teremtulajdonos által.

A függvénykomponensek állapotainak tárolása és menedzselése React Hook-ok segítségével történik, az életciklus metódusainak helyettesítése pedig a mobil komponensekhez hasonlóan *Effect Hook*-okkal, amely részletezve van a 2.3.2. alfejezetben.

2.2.3. Routing

A komponensek közötti belső navigáció a **React Router** [34] modul segítségével valósul meg, amely egy szabványos könyvtár a React útválasztáshoz. Lehetővé teszi a navigációt az alkalmazás különböző összetevőinek nézetei között, a böngésző URL címének módosítását, és szinkronban tartja a felhasználói felületet az URL-lel.

Az útvonalakat a *react-router-dom* függőségbe beépített *Route* komponens segítségével adhatjuk meg, amelynek props-ként megadhatjuk, hogy mi legyen az útvonal címe (path), és melyik komponensre navigáljon minket ez az útvonal (component). Például a **<Route path="/addRoom" component=AddRoom />** Route segítségével egy útvonalat tudunk megadni a */addRoom* URL-re, amely be fogja tölteni az *AddRoom* komponenst.

A React Routernek a szállításáról a *useHistory* horog gondoskodik, amely hozzáférést biztosít a router állapotához, és lehetővé teszi a navigációt a komponensek között.

```
return (
    <>
        <RoomDetailCard key={room._id} room={room} />

        {room.currentLocation && (
            <div style={styles.mapContainer}>
                <MapContainer currentLocation={room.currentLocation}/>
            </div>
        )}

        <ReservationDatePicker roomId={room._id} />
    </>
);
```

2. kódrészlet. Példa kódrészlet egy konkrét terem részleteinek a megjelenítéséről függvénykomponenseken belül

2.3. Mobil

A mobilalkamazás egy iOS-en, illetve Android-on is könnyen használható felhasználói felületet biztosít, háttérben kommunikálva az API szerverrel. A mobil-kliens Expo [7] segítségével van megvalósítva, mely a 3.3 fejezetben kerül bővebb bemutatásra. A forráskód négy fő csomagra osztható: *components*, *context*, *service* és *util*.

A **components** csomag a felületen megjelenő komponenseket tartalmazza, entitások szerint kategorizálva. Ennek köszönhetően átláthatóbb, könnyebben szerkeszthető a kód és lehetőség nyílik az elemek újrahasznosítására is.

A **context** csomag gyűjti össze az alkalmazásban több komponens által használt és módosított értékeket. Ezek Context [3] objektumokba helyezhetőek, majd feliratkozás által elérhetővé tehetőek a komponensek hierarchiájában.

A **service** csomagban található függvények felelősek a mobil-kliens és az API szerver közötti kommunikáció megvalósításáért. A komponensek ezek meghívásával adatokat kérhetnek le és küldhetnek el, a válaszok pedig számukra is felhasználható formában érkeznek.

A **util** csomag tartalmazza az alkalmazás működéséhez szükséges segédfüggvényeket, konstansokat, megelőzve ezzel az ismétlődő kódrészleteket és elírásokból származó hibákat.

2.3.1. Kommunikáció a szerverrel

A mobilalkalmazás és a központi szerver közötti kommunikáció REST [36] kérésekkel valósul meg. A HTTP kérések küldése és fogadása, a webalkalmazáshoz hasonlóan, Fetch API [11] segítségével történik. Az egyes entitásokhoz tartozó kéréseket megvalósító függvények a **service** csomag különböző fájljaiban találhatóak.

Ahogy azt a 3. kódrészlet is mutatja, a Fetch kérések alakja és paramétérezése megegyezik a 2.2 fejezetben ismertetett megoldásokkal. A kötelező elérési útvonal mellett megadhatóak kiegészítő opciók: itt specifikálható például a használni kivánt HTTP metódus, a *body* tartalma vagy az elküldendő fejlécek. A példakód esetében az alapértelmezett GET metódus POST-ra cserélődik, a *body*-ban egy *reservation* objektum lesz elküldve JSON formátumban, amit a *headers* mezőben tüntet fel, a bejentkezett felhasználó azonosítására szolgáló *token* kíséretében.

A Fetch kérésre érkező válasz egy *Promise*, melyet ebben az esetben is *async/await* szintaxis segítségével dolgoznak fel a service függvények. A visszatérített HTTP választ reprezentáló *Response* [12] objektum tulajdonságai, mint a visszatérített érték, fejlécek vagy a body, olvashatóak, illetve különböző metódusok állnak rendelkezésre a feldolgozás

megkönnyítése érdekében. A 3. kódrészletben bevárt aszinkron kérés elvárt eredménye egy JSON objektum, melyet a *response.json()* [37] metódus szolgáltat. Ugyanakkor a kapott válasz HTTP státuszkódja is ellenőrzött, hiba esetén a megjelenítést biztosító komponens jelzést kap a nem megfelelő végkimenetelről.

2.3.2. Komponensek

A React Native [32] technológiára épülő Expo [7] keretrendszer segítségével fejlesztett mobilalkalmazás megjelenített elemei tulajdonképpen *React komponensek*, melyek mindegyike egy meghatározott feladatot lát el (*single-responsibility principle* [40]), így minden komponens továbbiakra van bontva, amíg lehet.

A komponensek, akárcsak a web-kliens esetében, függvénykomponensekként vannak megírva, React Hook-okra [31] bízva a belső állapotok menedzselését. A .jsx [18] kiterjesztésű fájlokban megírt komponensek a *components* csomagban találhatóak. Ezen belül további csoportosítás valósul meg a megjelenített entitások/feladatkörök szerint. Például, a *rooms* alcsomag gyűjti össze a termekkel kapcsolatos elemeket, míg a *filter*-ben a keresést segítő szűrők vannak. A felosztásnak köszönhetően a forráskód átláthatóbb, könnyebben szerkeszthető, emellett az egyes komponensek akadály nélkül újrahasznosíthatóak, ezzel

```
export async function addReservation(reservation) {
    const token = await SecureStore.getItemAsync("token");
    const serverUrl =
        (await AsyncStorage.getItem("serverUrl")) || ENV.SERVER_URL;
    let response = await fetch(`${serverUrl}/api/reservations`, {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
            authorization: token,
        },
        body: JSON.stringify(reservation),
    });

    if (response.status === 500) {
        throw new Error(ALERTS.INSERTION_ERROR.MESSAGE);
    }

    return response.json();
}
```

3. kódrészlet. Példa a mobilalkalmazás által küldött kérésre Fetch API segítségével

megelőzve a kódismétlődéseket.

A 4. kódrészlet egy terem adatainak függvénykomponensből való megjelenítését mutatja be. A *View* konténerként viselkedik, további elemek adhatóak hozzá. A *RoomAttributes* komponensen belül jelennek meg a terem különböző tulajdonságai, ezek máshol is felhasználhatóak a kiemelés által. Ugyanakkor, feltételes kirajzolás is megvalósítható, ahogy az a *Flex* elemnél látható.

Az osztálykomponensek életciklus metódusainak helyettesítése érdekében a függvénykomponensek *Effect Hook*-okat [29] használnak: a *useEffect* hook tartalmazza a végrehajtandó kódrészletet és opcionálisan megadható egy lista a figyelendő változókkal. Emellett, a teljesítmény javításának érdekében megjelennek *useCallback* [35] hook-okba burkolt függvények. Így egy memorizált callback függvényt kapunk vissza eredményül, mely csak akkor változik, ha a megadott függőségek valamelyike módosult. Ez főleg olyan esetekben hasznos, amikor egy komponens függvényt ad tovább valamelyik beágyazott elemének.

```
return (
    <View style={styles.attributes}>
        <Text style={styles.boldText}>
            {ROOM_CARD_LABELS.NAME}:
            <Text style={styles.normalText}>{props.room.name}</Text>
        </Text>
        <RoomAttributes room={props.room} />
        {user && (
            <Flex style={styles.buttonFlex}>
                <Button
                    style={styles.button}
                    size="sm"
                    onPress={openReservationsByRooms}
                >
                    {ROOM_CARD_LABELS.RESERVATIONS}
                </Button>
            </Flex>
        )}
    </View>
);
```

4. kódrészlet. Példa egy terem adatainak megjelenítéséről függvénykomponensen belül

2.3.3. Adattárolás

A felhasználó bejelentkezéskor egy tokent kap az API szervertől. Ennek segítségével lesz beazonosítva a kérések feldolgozásakor, ezért fontos, hogy a token legyen eltárolva, a későbbi felhasználás érdekében. Erre biztonságos megoldást nyújt az Expo által biztosított SecureStore [8], mely lehetőséget teremt arra, hogy kulcs-érték párok enkriptált formában lokálisan lementhetőek legyenek az eszközre. iOS esetében ez a *keychain services* felhasználásával valósul meg, míg Androidon az értékek a *SharedPreferences*-ben tárolódnak.

3. Felhasznált technológiák és eszközök

Ebben a fejezetben bemutatásra kerülnek a felhasznált technológiák és eszközök, amelyek elősegítették és hatékonyabbá tették a fejlesztési folyamatot.

3.1. Szerveroldali technológiák

Node.js [24] A Node.js egy nyílt forráskódú és többplatformos JavaScript futási környezet, amely a Chrome V8 [42] JavaScript motorra épül, elsősorban webszerverek készítésére hozták létre. A Node.js alkamazások egyszerűen és gyorsan kezelik az adatokat, ezért ha az alkalmazás sok bemeneti és kimeneti (I/O) műveletet kell végezzen, akkor előnyös választás lehet. Számos beépített modult tartalmaz, amelyek megkönnyítik a fejlesztők dolgát. Továbbá külső függőségeket hozhatunk be a Yarn csomagkezelővel, amely a későbbiekben lesz részletezve.

Express.js [10] Az Express.js a Node.js keretrendszere, amelyet webalkalamzások és API szerverek építésére terveztek, robusztus szolgáltatáskészletet biztosít webes és mobilalkalamzásokhoz egyaránt. Megkönnyíti a HTTP-kérések és válaszok feldolgozását, valamint lehetővé teszi HTML oldalak dinamikus megjelenítését az argumentumok sablonoknak való átadása alapján.

MongoDB [21] A MongoDB nyílt forráskódú dokumentumorientált adatbázis szoftver, amely a NoSQL alapú adatbázisrendszerek közé tartozik. A MongoDB adatbázisok könnyen módosítható és bővíthető dokumentumokat tartalmaznak, nem ragaszkodnak szigorú sémákhoz, ezáltal gyakorlatilag bármilyen adatstruktúra modellezését és egyszerű kezelését lehetővé teszik. A dokumentumokat JSON-szerű formátumban, BSON-ben tárolja, amelyek különböző gyűjteményekbe struktúrálódnak, ezeknek köszönhetően nincs szükség például ORM keretrendszerekre (Object-Relational Mappers) ahhoz, hogy az adatokat objektumokba alakítsuk. Ugyancsak a MongoDB erősségeihez tartozik a geolokációs koordináták tárolása és elérése, ezáltal a SIRECO alkalmazás termeinek pontos elhelyezkedését könnyen tárolni tudja.

Mongoose [22] A Mongoose egy MongoDB objektummodellező JavaScript könyvtár, amely egy egyszerű, séma alapú megoldást kínál az alkalmazásadatok modellezésére. Aszinkron, Node.js környezetben való működésre tervezték. Általában a MongoDB szkriptek, validációk, konverziók írása idő-, és erőforrásigényes, sok boilerplate kódot igényelnek, ezeknek a helyettesítésére nyújt könnyű megoldást a Mongoose könyvtár.

3.2. Web technológiák

React.js [30] A web-kliens React segítségével van felépítve, amely egy ingyenes és nyílt forráskódú front-end JavaScript könyvtár, UI-komponenseken alapuló felhasználói felületek építésére szolgál. A React lehetővé teszi újrafelhasználható komponensek létrehozását, amelyek segítségével egy összetett felhasználói felület épülhet. A böngésző DOM-jának (Document Object Model) közvetlen manipulálása helyett létrehoz egy virtuális DOM-ot a memóriában, ahol elvégzi a szükséges változtatásokat, mielőtt módosítaná a böngésző DOM-ját. Adatváltozáskor csak a megfelelő komponensek lesznek frissítve, és újrarenderelve, azok a komponensek, amelyeken nem történt módosítás, változatlanok maradnak. A mobil klienshez hasonlóan az állapotmenedzselés, és a dinamikus tartalmak karbantartása React Hook-ok segítségével történik, amely részletezve van a 3.3. fejezetben. Ugyanúgy, mint a szerveroldalon, a különböző külső függőségek, és előre megírt könyvtárak behozatalát a yarn csomagkezelő valósítja meg.

React Router [34] Ahogy a 2.2.3. alfejezetben említve volt, a komponensek közötti navigáció React Router segítségével történik. Külső függőség behozására van szükség a *Route* komponens eléréséhez, amellyel különböző útvonalakat lehet definiálni, illetve meg lehet határozni, hogy melyik útvonalhoz milyen komponens tartozzon.

Ant Design [1] Az Ant Design egy kliensoldali tervező könyvtár, amely különböző előre megírt React komponenseket tartalmaz. Ezeknek köszönhetően egy esztétikus, felhasználóbarát felület hozható létre, többféle platformon is hasonló megjelenést adva az alkalmazásnak. Az *antd* külső függőség telepítésével férhetünk hozzá az Ant Design erőforrásaihoz és eszközeihez, amelyeket a hivatalos weboldalán [2] találunk meg.

3.3. Mobil technológiák

Expo [7] A mobil-kliens Expo segítségével van felépítve, amely platformfüggetlen *React* applikációk megvalósítására szolgáló keretrendszer. React Native [32] technológiára épül, kiegészítve azt fejlesztés, buildelés és kitelepítés során hasznos eszközökkel, gyakran használt könyvtárakkal (például azonnal működő Push értesítések). Bár a React Native-ben megírt kód is futtatható mind iOS, mind Android operációs rendszereken, ennek használatakor néha elkerülhetetlen natív modulok használata, illetve az iOS kliens buildeléséhez a fejlesztőnek egy Mac rendszerrel kell rendelkeznie. Ezzel ellentétben, az Expo esetében a build folyamat

teljes mértékben a keretrendszer feladata, az általa biztosított felhőben történik. Ugyanakkor, az alkalmazás elindításakor egy QR kód generálódik, ezzel és a mobilra letölthető *Expo Go* [6] applikációval az alkalmazás könnyen tesztelhető az azonos hálózaton lévő eszközökön, illetve szimulátorokon.

JSX [18] Az Expo, ahogyan a React Native is, Javascript alapú. A definiált komponensek JSX nyelvezetet használva íródtak, mely a Javascript kiterjesztése. Ezek natív UI elemekként renderelődnek Android és iOS platformokon.

React Hooks [31] Ahogyan a 2.3. fejezetben említésre került, függvénykomponensekkel valósul meg a felhasználói felület. Az osztályok által biztosított állapot hozzáadására ezekhez a React Hooks nyújt megoldást. Ennek segítségével állapotváltozók és őket frissítő függvények deklarálhatóak a *useState()* parancs meghívásával.

React Navigation [33] A komponensek közötti navigáció React Navigation segítségével valósul meg. Ez különböző navigációs módszereket tesz lehetővé, melyek egymásba is ágyazhatóak. A projekt mobil-kliensének esetében a *Drawer navigation* szolgál alapként, mely egy oldalról kihúzható menü segítségével egyszerű váltást biztosít az oldalak között. Az ezen belül megjelenített komponensek egy része viszont belső navigációra *Stack navigator*-t használ, mely nyilvántartja a váltások sorrendjét, így könnyű az előzőre való visszalépés.

Native Base [23] Az alkalmazás megjelenésének egységességét és stílusjegyeit a Native Base design könyvtár biztosítja. Mivel React Native elemekre épül, minden platformon megjeleníthető komponenseket biztosít, melyek könnyen személyreszabhatóak.

3.4. Eszközök és módszerek

Visual Studio Code [43] A Visual Studio Code a Microsoft egyik fejlesztői környezete, beépített *Javascript* és *Node.js* támogatással, így a szerver, a web- és a mobilalkalmazás fejlesztése alatt is segítségül szolgált. Emellett, a verziókövetés folyamatát is leegyszerűsítte, a *Source Control Management* funkcionalitásnak köszönhetően.

MongoDB Compass [20] A Compass egy interaktív eszköz, mely felhasználói felületet biztosít MongoDB adatbázisok létrehozására, az ezekben tárolt adatok lekérdezésére és kezelésére.

Yarn [44] A Yarn egy csomagkezelő rendszer, a Facebook a Node.js JavaScript futási környezethez fejlesztette ki, amely sebességet, stabilitást, és biztonságot biztosít a Node Package Manager (npm) alternatívájaként. Lehetővé teszi a felhasználók számára a rendszerleíró adatbázisban elérhető JavaScript modulok használatát, az elérhető csomagok pedig a yarn weboldalán [45] keresztül böngészhetőek és kereshetőek. A Yarn beépülő modullal rendelkezik, nemcsak néhány alapértelmezés szerint használható bővítményt tartalmaz, hanem a felhasználók saját beépülő moduljaikat is elérhetővé tehetik. A különböző függőségek és modulok, valamint a projekt metaadatai, a *package.json* leírófájlban találhatóak, ennek segítségével tudjuk telepíteni az egyes függőségeket. A szerver, a web és mobil kliensek esetében is ez az eszköz segítette a függőségek kezelését.

Git [14] A Git egy elterjedt verziókövető rendszer, mely a kódbázis különböző állapotait tárolja. Ezáltal az egyes verziók között történt változások könnyen visszanézhetőek és, szükség esetén, lehetséges egy korábbi változatra való visszaállás. A fejlesztés alatt a *git-flow* [15] által előírt konvenciók kerültek betartásra: a *develop* ág tartalmazta az alkalmazás aktuális állapotát, az új funkcionalitások kidolgozására pedig új, dedikált ágak jöttek létre, melyek ellenőrzés után kerültek be a *develop* ágra. A SIRECO projekt három tárolóval rendelkezik: *i-backend*, *i-mobile* és *i-web*.

GitLab [17] A GitLab egy *DevOps* platform, mely megkönnyíti a git tárolók karbantartását, egyszerű felhasználói felületet biztosítva a változások követésére, a csapattársak által írt kódok ellenőrzésére. Emellett, rendelkezik egy táblával, melynek segítségével a kitűzött feladatok kategóriákba rendezhetőek, megfelelő cimkék megadásával, és egy *kanban board*-on kezelhetőek.

GitLab Continous Integration (CI) [16] A CI a GitLab beépített folyamatos integrációs szolgáltatása, mely az új részek helyes integrációját biztosítja. Lehetőséget biztosít *pipeline*-ok futtatására minden *push* művelet esetében. Ezek segítségével valósultak meg a statikus kódellenőrzés (*eslint* [5] és *prettier* [28] által), és a build folyamatok. Abban az esetben, ha valamelyik végrehajtása sikertelen, a fejlesztő értesítést kap erről.

Scrum [39] A fejlesztés a Scrum agilis szoftverfejlesztői keretrendszer szabályai szerint történt. A fejlesztés folyamata *sprint*ekre bomlott, melyek időtartama kezdetben egy hét, majd a *Csoportos Projekt* egyetemi tárgy alatt két hét volt. Ez egy tervezési találkozóval (*planning*) indult, melynek keretein belül megbeszelésre kerültek a sprint végéig megvalósításra szánt

funkcionalitások, a GitLab által biztosított tábla segítségével rendszerezve. A sprint alatt napi szintű rövid megbeszélesekre (*daily*) is sor került, ahol a csapat tagjai haladásukat és elakadásaikat ismertethették. A sprintet egy *demo* zárta, a sikeresen elkészített funkcionalitások bemutatásával.

4. A SIRECO alkalmazás működése

A SIRECO szoftverrendszer funkcionalitásai a felhasználók számára a web- és mobilalkalmazás felületein elérhetőek, ezek használata kerül bemutatásra ebben a fejezetben. A működést néhány képernyőfotó is szemlélteti.

4.1. A webes felület használata

A webalkalmazás megnyitásakor a 7. ábrán látható kezdőoldal fogad, kezdetben nincs szükség bejelentkezésre, viszont csak korlátozott számú funkcionalitás érhető el.

Az oldalon csak az elérhető termek jelennek meg a fontosabb információikkal, a felhasználónak csak böngészni van joga közöttük. Ennek a böngészésnek a megkönnyítésére szolgálnak a navigációs sáv alatt elhelyezkedő szűrő mezők, ezek segítségével hatékonyabban és gyorsabban megtalálhatóak a felhasználó igényeit leginkább kielégítő termek. Lehetőség van szűrni intézmény, név, város, ország, teremtípus, felszereltség alapján, valamint ezek alatt található egy csúszka is, amelyet egy bizonyos intervallumra beállítva, csak azok a termek fognak megjelenni, amelyeknek a befogadóképessége ebbe az intervallumba esik. Továbbá, a szűrőmezők mellett található egy *Search on map* gomb is, ennek lenyomásával egy térképes oldal jelenik meg, amelyen meg van jelölve az összes terem pontos lokációja.

Az oldal tetején található a navigációs sáv, a felhasználó itt váltogathat az általa elérhető oldalak között. Ha a fent említett lehetőségeken kívül további funkcionalitásokhoz is hozzá

7. ábra. Az alkalmazás kezdőoldala, amely az oldal megnyitásakor fogad

szeretne férni, akkor regisztrációra, vagy bejelentkezésre van szüksége.

Bejelentkezés után az oldal jobb felső sarkában látható lesz az aktuálisan bejelentkezett felhasználó felhasználóneve, és eltűnnek a regisztrációs menüpontok, illetve újabbak lesznek elérhetőek. Megjelenik az *Add Institute* menüpont, amely alatt a felhasználó kérelmezheti az adminisztrátortól intézményének hozzáadását. Azért van szükség az adminisztrátor jóváhagyására, hogy az alkalmazás ne teljen meg spam tartalmakkal, ne tudjon valaki beszúrni olyan termeket, amelyek valójában nem is léteznek. Emellett megjelenik egy *My reservations* menüpont is, amely alatt a felhasználó foglalásai lesznek listázva, valamint lehetőség van a függőben levő foglalások dátumának vagy időpontjának módosítására, vagy abban az esetben, ha már nem aktuális a foglalás, annak törlésére is.

Továbbá megjelenik minden terem mellett egy *Details* gomb, amely által megjeleníthetőek a terem részletes adatai (lásd: 8 ábrán). Ez az oldal legfelül ugyanazt a kártya komponenst jeleníti meg, amely a termet reprezentálja. Emellett, ahogy a 8 ábra is mutatja, tartalmazza a terem leírását, ahol a teremtulajdonosnak lehetősége van megadni a kapcsolódó specifikus információkat.

Megjelenik egy térkép is, amelyen meg van jelölve a terem pontos helye, valamint egy naptár, ahol az aktuális terem terheltségétől függően jelezve vannak a szabad, részlegesen foglalt, és a teljesen foglalt napok. Ahogy a 8. ábrán is látszik, a narancssárga színű napok teljesen foglaltak, a sárga színűek pedig részlegesen, vagyis van már valakinek foglalása arra

8. ábra. Részletes információk egy konkrét teremről

a napra, viszont létezik még szabad időpontja a teremnek. A felhasználónak lehetősége van foglalásra, ehhez megjelenik egy dátumválasztó komponens, amelyben meg tudja adni a kezdő és záró időpontokat, és a *Reserve* gomb lenyomásával létrejön a foglalás. Lefele haladva az oldalon, a bejelentkezett felhasználó megtekintheti a teremhez tartozó foglalásait, és azok állapotát, hogy elfogadott, elutasított, vagy még várakozó állapotban vannak. Az oldal alján képek jelennek meg az adott teremről, amelyek segíthetik a felhasználót a választásban.

Ha egy felhasználó kérelmezte az adminisztrátortól intézményének bevezetését a fent említett menüpont alatt, és ez elfogadásra került, intézménytulajdonossá válik. Ettől a ponttól kezdve számára újabb menüpontok, és funkcionalitások jelennek meg. Az *Add room* menü alatt lehetősége van termeket hozzáadni az intézményéhez, ekkor bizonyos információkat kötelezően meg kell adnia, ahogy azt a 9a. ábra is mutatja.

Emellett megjelenik az intézménytulajdonosok számára egy *My rooms* menüpont is, ahol a saját intézményéhez tartozó termek lesznek listázva. Ha a saját termüknek a *Details* oldalára lépnek, lehetőségük nyílik feltölteni borítóképet a teremhez, vagy ha már van, akkor lecserélni azt, extra képeket feltölteni, és talán ami a legfontosabb, a teremhez tartozó foglalásokat is tudják menedzselni. Ezeket a funkcionalitásokat szemlélteti a 9b. ábra. Az intézménytulajdonosok számára a foglalások listázása mellett a *Pendig* státusszal rendelkező foglalási kérelmekhez tartozni fog két gomb is, amelyekkel lehetőségük van elfogadni, vagy

(a) Új terem felvezetéséhez szükséges űrlap

(b) Foglalások visszaigazolása, képek feltöltésének lehetősége az intézménytulajdonos számára

9. ábra

elutasítani azokat, annak függvényében, hogy melyik foglalás felel meg leginkább az intézmény igényeinek, lehetőségeinek.

Továbbá, a SIRECO alkalmazás rendelkezik adminisztrátor funkcionalitással is, ebbe a szerepkörbe tartozik a spam tartalmak szűrése. Az admin joggal rendelkező felhasználó számára nem jelennek meg a fentebb említett hagyományos menüpontok, viszont megjelenik egy *Institute requests* menüpont, amely alatt listázódnak az intézménykérelmek, ahogy azt a 10. ábra is mutatja. Addig a pontig, amíg az admin nem fogadja el egy adott intézménynek a hozzáadását, termeket sem lehet felvezetni az adott intézményhez. Minden intézmény alatt megjelenik az *Accept* és a *Decline* gomb, amelyek segítségével dönthet a beszúrandó intézmények jóváhagyásáról. A 10. ábrán továbbá látszik az is, hogy az adminisztrátor elfogadott egy intézménykérelmet, mivel mindig érkezik értesítés a művelet sikerességéről vagy sikertelenségéről.

10. ábra. Intézménykérelmek az adminisztrátor számára

4.2. A mobil alkalmazás használata

A mobil alkalmazás megnyitásakor, a webes felülethez hasonlóan, a bérelhető termek listája fogadja a felhasználót. Ez megjeleníti a termek borítóképét és fontosabb információit, ahogy azt a 11a. ábra szemlélteti. A bal felső sarokban elhelyezkedő szűrő ikon lenyomásával lehetőség nyílik különböző tulajdonságok szerinti keresésre, leszűkítve a találatokat a feltételeknek megfelelőekre. A szűrőket tartalmazó felugró *actionsheet* a 11b. ábrán látható. Ugyancsak a bal felső sarokban lévő térkép ikon vezet a termeket térképen megjelenítő oldalra, így láthatóvá válik a pontos elhelyezkedésük (lásd: 11c. ábra). Erre az oldalra navigálva a bal felső sarokban egy listázási ikon tűnik fel, ezzel váltható vissza a nézet. A menü ikon a jobb felső sarokban található, ennek kiválasztásával jelenik meg a navigációs menü, mely a felhasználó szerepkörének függvényében különböző menüpontokat tartalmazhat. A 12a. ábra az egyszerű felhasználó számára megjelenített lehetőségeket ábrázolja.

A rendszerbe már bejelentkezett felhasználók az egyes termek részleteit megjelenítő oldalakat is elérhetik. Ide kétféleképpen juthatnak el: a listázó oldalon látható *Details* gombok segítségével (ez bejelentkezés után minden terem kártyájának alján jelenik meg) vagy a térképes nézetben a megfelelő pinek címkéinek lenyomásával. Ezen az oldalon, a listában feltüntetett

(a) Az alkalmazás kezdőoldala vendég számára

(b) A termek szűrési lehetőségei

(c) Termek térképalapú megjelenítése

11. ábra

adatok mellett, további jellemzők is elérhetőek, egy képgaléria és a pontos elhelyezkedést megjelenítő térkép kíséretében (lásd: 12b. ábra). A felhasználók számára ezen az oldalon nyílik lehetőség a kiválasztott terem lefoglalására is. Naptár segítségével megjelennek a már lefoglalt dátumok, így a felhasználó könnyen megelőzheti a kérelmének későbbi visszautasítását a terem foglaltsága miatt (narancssárga színnel a részben, míg pirossal a teljesen foglalt napok jelennek meg). A naptár alatti alapértelmezetten megadott dátumok lenyomásával választhatja ki a kívánt időpontokat, majd a *Reserve* gomb által küldheti el a foglalást (lásd: 12c. ábra). Amint ez megtörtént, a foglalás az oldal alján lévő listában is megjelenik, a felhasználó által az adott teremre vonatkozó foglalások mellett.

A felhasználó megtekintheti foglalásait az erre a célra létrehozott *My reservations* oldalon, az azonos elnevezésű menüpont kiválasztásával. Ahogy a 13a. ábrán látszik, itt megjelennek az egyes foglalások adatai: a foglalt terem neve, a pontos időpontok, illetve a foglalás állapota. Kezdetben minden elküldött foglalás visszaigazolásra vár, ezt jelzi a *Pending* állapot. A még függőben lévő foglalások időpontjai az *Edit* gomb kiválasztásával módosíthatóak is. Amint a terem tulajdonosa elfogadja vagy elutasítja azt, az állapot *Accepted* vagy *Declined* lesz. Ezután már nincs lehetőség változtatásokra, csak a visszavonás lehetősége adott.

A foglalói szerepkörrel rendelkező felhasználók az *Add Institute* oldalon bevezethetik saját

(a) Az alkalmazás navigációs menüje

(b) Terem részleteit megjelenítő oldal

(c) Teremfoglalási lehetőség megvalósítása

12. ábra

intézményüket is. Itt meg kell adniuk a bevezetni kívánt intézmény nevét és opcionálisan egy borítóképet is. A foglalási kérelmekhez hasonlóan kezdetben ez a kérés is függő állapotba kerül, az adminisztrátor visszaigazolására vár. Ha sikeresen bekerül a rendszer intézményei közé, a felhasználó intézménytulajdonossá válik és meghirdetheti termeit az *Add room* oldalon található kérdőív kitöltésével (lásd: 13b. ábra). Itt megadhatóak a terem jellemzői, kiválasztható a pontos elhelyezkedése a térképen, és feltölthető egy borítókép. Az elküldés után a terem már elérhető a főoldali listában. Amennyiben a tulajdonos a részleteket megjelenítő oldalra lép, az egyszerű felhasználókkal ellentétben, nem foglalásra van lehetősége, hanem módosíthatja az aktuális borítóképet és új képeket tölthet fel a galériába (lásd: 13c. ábra).

A teremtulajdonos szerepkörű felhasználók számára elérhető a *My rooms* oldal is (lásd: 14a. ábra). Itt a felhasználó intézményeihez tartozó termek listája jelenik meg a főoldalon is látható fontosabb adatokkal. A kártyák jobb alsó sarkában elhelyezkedő *Show reservations* gomb az adott teremhez intézett foglalások listájához vezet. Itt minden foglalás esetében megjelenik annak tulajdonosának neve, az időpontjai és az állapota. Amennyiben az állapot még függő, a tulajdonos visszaigazolhatja azt, kiválasztva az *Accept* és *Decline* lehetőségek közül a megfelelőt. Ezt a 14b. ábra szemlélteti.

Az adminisztrátor felelőssége a felhasználók intézménytulajdonosi kérelmeinek az

(a) Felhasználó foglalásainak listája

(b) Terem hozzáadásakor kitöltendő form

(c) Tulajdonos lehetőségei a részleteket megjelenítő oldalon

13. ábra

elbírálása. Ezek listáját az erre létrheozott *Institute requests* oldalon tudja megtekinteni, ahogy azt a 14c. ábra mutatja. A kártyákon megjelenik az intézmény neve és borítóképe, illetve két gomb: *Accept* és *Decline*, melyek segítségével visszaigazolhatja a kérelmeket. Számára a mobilalkalmazás más funkcionalitásai, a termek listázását kivéve, nem elérhetőek.

(a) Teremtulajdonos termeinek listája

(b) Tulajdonos adott terméhez intézett foglalások listája

(c) Intézményregisztrálási kérelmek kezelése

14. ábra

Következtetések és továbbfejlesztési lehetőségek

A SIRECO projekt kereteiben létrejött szoftverrendszer a felhasználók számára mind webes felületen, mind mobilon elérhető kliensalkalmazásokat biztosít. Ezek segítségével a konferenciákat, nagyobb rendezvényeket szervező felhasználók böngészhetnek a bérlésre meghirdetett termek között, megtekinthetik azok részletes jellemzőit, elhelyezkedésüket, és, amint megtalálták a számukra legmegfelelőbb helyiségeket, könnyen lefoglalhatják azokat. Azok a felhasználók, akik valamilyen intézménynek vagy cégnek a képviselői, kérhetik az adott intézmény bevezetését a rendszerbe. Miután az adminisztrátor jóváhagyta kérésüket, a hozzájuk tartozó bérelhető konferencia-, előadó- vagy dísztermeket meghirdethetik.

A fejlesztés alatt több elképzelés született a szoftverrendszer funkcionalitásait illetően, ezek egy részét sikerült megvalósítani. A fennmaradó ötletek képezik a továbbfejlesztési lehetőségeket, melyek közül néhány példa:

- A rendszer kibővítése termektől különálló, bérelhető eszközök meghirdetésének lehetőségével, melyek szükségesek lehetnek nagyobb rendezvények szervezésekor (például ledfalak, vetítők, hangosítás stb.).
- Az alkalmazásokon keresztül lefoglalt dátumok átvezetése az intézmények belső foglalási rendszerének naptárába, ezzel megkönnyítve a tulajdonosok dolgát az adatszinkronizálás szempontjából.
- Az intézménytulajdonos számára jogot adni más felhasználók hozzáadására az intézményhez, ezzel könnyítve a nagyobb intézmények adatainak kezelését.
- A mobil alkalmazás esetében értesítési rendszer bevezetése, mely által a felhasználók értesülnének foglalásaik visszaigazolásáról, a teremtulajdonosok pedig az újonnan beérkezett foglalásokról.
- Alkalmazáson belüli fizetési lehetőség bevezetése, amennyiben egy foglalás elfogadásra került.

Hivatkozások

[1] *Ant Design hivatalos dokumentáció*. URL: https://ant.design/docs/react/introduce (elérés dátuma: 2022. ápr. 12.)

[2] *Ant Design komponensek*. URL: https://ant.design/components/overview/ (elérés dátuma: 2022. ápr. 12.)

[3] *Context in React*. URL: https://hu.reactjs.org/docs/context.html (elérés dátuma: 2022. ápr. 5.)

[4] *Data Transfer Object*. URL: https://martinfowler.com/eaaCatalog/dataTransferObject.html (elérés dátuma: 2022. ápr. 2.)

[5] *Eslint hivatalos weboldala*. URL: https://eslint.org/ (elérés dátuma: 2022. ápr. 12.)

[6] *Expo Go applikáció*. URL: https://expo.dev/client (elérés dátuma: 2022. ápr. 11.)

[7] *Expo hivatalos dokumentáció*. URL: https://docs.expo.dev/ (elérés dátuma: 2022. márc. 23.)

[8] *Expo SecureStore*. URL: https://docs.expo.dev/versions/latest/sdk/securestore/ (elérés dátuma: 2022. ápr. 7.)

[9] *Express-Router*. URL: https://expressjs.com/en/guide/routing.html#express-router (elérés dátuma: 2022. ápr. 2.)

[10] *Express.js hivatalos dokumentáció*. URL: https://expressjs.com/ (elérés dátuma: 2022. márc. 23.)

[11] *Fetch API*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (elérés dátuma: 2022. ápr. 4.)

[12] *Fetch API Response*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Response (elérés dátuma: 2022. ápr. 7.)

[13] *Firebase Cloud Storage hivatalos dokumentáció*. URL: https://firebase.google.com/docs/storage (elérés dátuma: 2022. márc. 23.)

[14] *Git hivatalos dokumentációja*. URL: https://git-scm.com/doc (elérés dátuma: 2022. ápr. 11.)

[15] *Git workflow*. URL: https://about.gitlab.com/topics/version-control/what-is-git-workflow/ (elérés dátuma: 2022. ápr. 11.)

[16] *GitLab CI*. URL: https://docs.gitlab.com/ee/ci/introduction/index.html#continuous-integration (elérés dátuma: 2022. ápr. 12.)

[17] *GitLab hivatalos dokumentáció*. URL: https://docs.gitlab.com/ (elérés dátuma: 2022. ápr. 12.)

[18] *Introducing in JSX*. URL: https://hu.reactjs.org/docs/introducing-jsx.html (elérés dátuma: 2022. ápr. 6.)

[19] *JSON Web Token*. URL: https://jwt.io/introduction (elérés dátuma: 2022. ápr. 3.)

[20] *MongoDB Compass*. URL: https://www.mongodb.com/products/compass (elérés dátuma: 2022. ápr. 11.)

[21] *MongoDB hivatalos dokumentáció*. URL: https://www.mongodb.com/docs/ (elérés dátuma: 2022. márc. 23.)

[22] *Mongoose hivatalos dokumentáció*. URL: https://mongoosejs.com/docs/api.html (elérés dátuma: 2022. márc. 23.)

[23] *Native Base*. URL: https://nativebase.io/ (elérés dátuma: 2022. ápr. 11.)

[24] *Node.js hivatalos dokumentáció*. URL: https://nodejs.org/en/docs/ (elérés dátuma: 2022. márc. 23.)

[25] *OAuth2*. URL: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2 (elérés dátuma: 2022. ápr. 3.)

[26] *Oauth2 Architecture*. URL: https://docs.oracle.com/cd/E82085_01/160027/JOS%20Implementation%20Guide/Output/oauth.htm (elérés dátuma: 2022. ápr. 7.)

[27] *OpenStreetMap*. URL: https://www.openstreetmap.org/#map=5/51.500/-0.100 (elérés dátuma: 2022. ápr. 7.)

[28] *Prettier hivatalos dokumentáció*. URL: https://prettier.io/docs/en/ (elérés dátuma: 2022. ápr. 12.)

[29] *React Effect Hook*. URL: https://reactjs.org/docs/hooks-effect.html (elérés dátuma: 2022. ápr. 7.)

[30] *React hivatalos dokumentáció*. URL: https://reactjs.org/ (elérés dátuma: 2022. márc. 23.)

[31] *React Hooks*. URL: https://hu.reactjs.org/docs/hooks-intro.html (elérés dátuma: 2022. ápr. 6.)

[32] *React Native hivatalos dokumentáció*. URL: https://reactnative.dev/docs/getting-started (elérés dátuma: 2022. ápr. 7.)

[33] *React Navigation*. URL: https://reactnavigation.org/docs/getting-started (elérés dátuma: 2022. ápr. 11.)

[34] *React Router*. URL: https://www.geeksforgeeks.org/reactjs-router/ (elérés dátuma: 2022. ápr. 7.)

[35] *React: useCallback*. URL: https://hu.reactjs.org/docs/hooks-reference.html#usecallback (elérés dátuma: 2022. ápr. 7.)

[36] *Representational State Transfer*. URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (elérés dátuma: 2022. ápr. 4.)

[37] *Response.json()*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Response/json (elérés dátuma: 2022. ápr. 6.)

[38] Mark Richards. *Software Architecture Patterns: Understanding Common Architecture Patterns and when to Use Them*. O'Reilly Media, 2015.

[39] *Scrum hivatalos dokumentáció*. URL: https://www.scrum.org/resources/what-is-scrum (elérés dátuma: 2022. ápr. 12.)

[40] *Single-Responsability Principle*. URL: https://www.baeldung.com/java-single-responsibility-principle (elérés dátuma: 2022. ápr. 6.)

[41] *Single-Responsability Principle*. URL: https://www.baeldung.com/java-single-responsibility-principle (elérés dátuma: 2022. ápr. 6.)

[42] *V8 JavaScript motor*. URL: https://nodejs.org/en/docs/ (elérés dátuma: 2022. márc. 23.)

[43] *Visual Studio Code*. URL: https://code.visualstudio.com/docs (elérés dátuma: 2022. ápr. 11.)

[44] *Yarn hivatalos dokumentáció*. URL: https://yarnpkg.com/getting-started (elérés dátuma: 2022. ápr. 12.)

[45] *Yarn hivatalos oldala*. URL: https://yarnpkg.com/ (elérés dátuma: 2022. ápr. 12.)