ETDK-dolgozat

Mihály Dénes

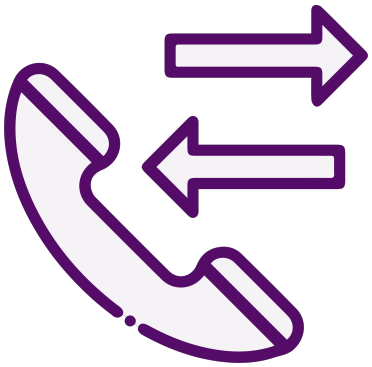
Székely Szilárd-Lehel

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

Kolozsvár, 2022. május 19–22.

Helpline

Órarend alapú hívásátirányítást megvalósító alkalmazás



Szerzők:

Mihály Dénes

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

Székely Szilárd-Lehel

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

Témavezetők:

dr. Simon Károly, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Kiss Alpár, szoftverfejlesztő,

Codespring

Séra Barna, szoftverfejlesztő,

Codespring

Simó Attila, szoftverfejlesztő,

Codespring

Kivonat

A kisebb vállalkozások és önkéntes szervezetek esetében jellemzően gondot jelent a központi telefonra beérkező hívások célbajuttatása a megfelelő alkalmazottakhoz. Ezt a problémát sok esetben körülményes megoldásokkal kényszerűlnek kezelni, mivel a szükséges költségvetés hiányában nincs lehetőségük a piacon található drágább rendszereket igénybe venni.

A Helpline projekt célja egy olyan költséghatékonyan üzembe helyezhető és fenntartható rendszer elkészítése, ami automatikusan átirányítja a beérkező hívásokat egy időbeosztás alapján.

Az alkalmazás két fő része közé tartozik egy webes felület, ahol a szervezetek vezetői és az alkalmazottak közösen szerkeszthetik órarendjüket egy interaktív naptáron. A szervezet központi telefonjára kitelepített Android alkalmazás, kizárólag a készülék alapfunkcionalitásait használva irányítja át a beérkező hívásokat az időbeosztás szerint éppen aktív alkalmazott telefonjára.

A dolgozat bemutatja a rendszer részletes működését, a használt technológiákat és részletezi a funkcionalitásokat.

Tartalomjegyzék

Bevezető **1**

1. A Helpline projekt **2**

1.1. Piackutatás 2

1.2. Szerepkörök és funkcionalitások 3

1.2.1. Vendég felhasználó 4

1.2.2. Regisztrált felhasználó 4

1.3. Architektúra 5

2. A szerver **7**

2.1. Technológia 7

2.2. Adatmodell 7

2.3. API 8

2.4. Hitelesítés és engedélyezés 10

3. A webalkalmazás **12**

3.1. Technológia 12

3.2. Adatok megjelenítése 12

3.3. Kommunikáció a szerverrel 13

4. A mobilalkalmazás **14**

4.1. Hívásátirányítás MMI kódokkal 14

4.2. Hívásnaplózás 15

5. Fejlesztési folyamat és eszköztár **16**

5.1. Szoftverfejlesztés SCRUM segítségével 16

5.2. Verziókövetés 16

5.3. CI/CD 17

6. A Helpline működése **18**

6.1. Webes kliens 18

6.2. Mobilalkalmazás 22

Következtetések és továbbfejlesztési lehetőségek **24**

Bevezető

A központi telefonra beérkező hívások célbajuttatása a megfelelő alkalmazottakhoz jellemzően gondot okoz a kisebb vállalkozások és önkéntes szervezetek esetében. Költségvetés hiányában nincs lehetőségük a piacon található drágább rendszereket igénybe venni, ezért körülményes megoldásokhoz kell folyamodniuk.

A Helpline projekt ezt a problémát hivatott megoldani egy olyan költséghatékonyan üzembe helyezhető és fenntartható rendszer elkészítésével, ami időbeosztás alapján automatikusan átirányítja a beérkező hívásokat. A szervezetek vezetőinek és az alkalmazottaknak lehetőséget biztosít az időbeosztások szerkesztésére, hozzájárulva a munkafolyamatok hatékonyabb megszervezéséhez.

A projekt elindítása az Erdélyi Telefonos Lelkisegély-Szolgálat felkérésének alapján történt, akik jelezték, hogy szükségük lenne egy olyan hívásátirányítást megvalósító rendszerre, amely a hozzájuk hasonló kisebb költségvetési keretekkel rendelkező szervezetek számára is elérhető.

A dolgozat 1. fejezete tömören ismerteti a projekt működését, a szerepköröket, valamint előnyeit a piacon már megtalálható, hasonló termékekkel szemben. Az alkalmazást alrendszerekre bontja, melyet a további fejezetek részletesen tárgyalnak.

A következő három fejezet a három fő komponenst és a közöttük levő kommunikációt mutatja be. A 2. fejezet témája az API szerver, a felhasznált adatmodellek, a perzisztencia és a biztonság. A weboldal által felhasznált technológiákat, a szerverrel való kommunikációt, valamint az adatok megjelenítésének részleteit a 3. fejezet ismerteti. A 4. fejezet a mobilalkalmazás működésébe vezeti be az olvasót, a hívásátirányítást és a naplózást részletezve.

Az 5. fejezetben a fejlesztés során felhasznált fontosabb eszközök kerülnek bemutatásra. Az ezt követő fejezet az alkalmazás működését ismerteti az olvasóval. Zárásként a következtetésekről és továbbfejlesztési lehetőségekről esik szó.

A projekt 2021 nyarán indult el a U-Hub Mentorprogram keretein belül a Codespring koordinálásával, majd folytatódott a fejlesztése az egyetemi "csoportos projekt" tantárgy során. A csapat ez idő alatt új tagokkal egészült ki, név szerint: Bán Éden-Aida, Bánto-Laczi Klára, Nagy Norbert-Botond és Salamon Csongor.

Köszöneti illeti a témavezetőt, dr. Simon Károlyt, Tüzes-Bölöni Kincsőt, a webalkalmazás design-jáért és mentorainkat: Kiss Alpárt, Séra Barnát és Simó Attilát, akik szakmai tudásukkal jelentősen hozzájárultak a projekt fejlesztéséhez.

1. A Helpline projekt

A Helpline projekt célja egy olyan költséghatékony rendszer fejlesztése, amely időbeosztás alapján automatikusan átirányítja a beérkező telefonhívásokat az éppen aktív alkalmazott telefonjára. A rendszer egy webes felületből és egy mobilalkalmazásból tevődik össze. A felhasználók a webalkalmazáson keresztül szerkeszthetik időbeosztásaikat, valamint a szervezetek vezetői is itt láthatják el adminisztrációs feladataikat. A szervezet központi telefonjára kitelepített Android alkalmazás a hívásátirányításért felelős, a készülék alapfunkcionalitásait használva.

A fejezet a fejlesztés motivációját tárgyalja, majd felhasználói szerepkörökre lebontva ismerteti a rendszer működését és architektúráját.

1.1. Piackutatás

A projekt által felvetett probléma az az ügyfelek hívásainak fogadása vagy átirányítása, amely nélkülözhetetlen feladat a nagyobb cégek, szervezetek életében. A probléma jelentőségéből adódóan a nagyobb cégek saját erőforrásokat - belső fejlesztésű rendszereket, alkalmazottakat - különítenek el a feladathoz. A saját rendszerek mellett a piacon számos *call-center* alkalmazás igénybe vehető, melyek a hívások átirányítása mellett más funkciókat is nyújtanak. A következőkben a piacon található népszerűbb termékek fő jellegzetességei kerülnek bemutatásra, összehasonlítva a Helpline nyújtotta szolgáltatásokkal.

Ringover

A Ringover [27] jó minőségű kapcsolattartást biztosít cégek és vásárlóik között. A cég terméke havi 24 dolláros előfizetésért korlátlan hívást engedélyez 110 országba, támogatja a HD minőségű videóhívásokat és a hívások rögzítését. A felsorolt funkcionalitások mellett az előfizető átirányíthatja a beérkező hívásokat egy másik eszközre vagy telefonszámra. A rendszer beállításakor az adminisztrátor megadhatja, hogy csak bizonyos időintervallumban történjen meg az átirányítás.

Dialpad

A Dialpad [5] felhasználónként havi 20 dollárért nyújt teret a céges kommunikációkhoz. Az előfizetés magába foglalja a céges telefonrendszert, üzenetküldési és meeting alkalmazást,

beépített statisztikákat, valamint Google Workspace és Microsoft 365 integrációt. A céges telefonrendszer funkcionalitásai közé tartozik a hívásátirányítás, ami egyidőben maximum 5 átirányitási célpont megadását támogatja, ezért több előfizetésre is szükség van a funkcionalitás kihasználásához.

SIP szerverek

A kész alkalmazások mellett, egy megoldást nyújthatnak a szervezetek belső hálózatába beépített SIP (Session Initiation Protocol) szerverek. Ez a megközelítés több szempontból is előnyös lehet egy szervezet számára: a hívások kezelése sokkal flexibilisebbé válik, könnyen átirányíthatóak egyik eszközről a másikra, csökken a felhasznált sávszélesség és lehetővé teszi a VoIP (Voice Over Internet Protocol) alapú kommunikációt az eszközök között. Habár a SIP szerveren alapuló rendszerek használata jelentős előnyökkel szolgál, beüzemelésük költséges, és karbantartásukat, illetve kezelésüket általában a szervezet IT részlege végzi. A kisebb szervezetek, mint például a projekt alapötletét szolgáltató Erdélyi Telefonos Lelkisegély-Szolgálat, nem engedhetnek meg maguknak ilyen szintű beruházást. Ugyanakkor nem is tudnák kihasználni a SIP szerverek által nyújtott szolgáltatások zömét, lévén, hogy csak a beosztás alapú hívásátirányítás szükséges számukra.

Összegzés

A felsorolt példák olyan termékcsomagokat kínálnak, amelyeknek része a hívásátirányítás, így a szervezetek olyan szolgáltatásokért is fizetni kényszerülnek, amelyre valójában lehet, hogy nincs is szükségük. Ezzel ellentétben, a Helpline kimondottan a hívásátirányítást megoldani kívánó szervezeteket célozza termékével, ezáltal pénzügyi megtakarítást eredményez. A Helpline előnye a többi termékkel szemben, hogy az alkalmazottak is kezelhetik a hívásátirányítást az interaktív naptáron keresztül, így a működés nincs egyetlen adminisztrátor felhasználóhoz kötve. Mindezek mellett a hívásátirányítás feltételei rugalmasak, és korlátlan telefonszámra alkalmazhatóak egyetlen előfizetéssel.

1.2. Szerepkörök és funkcionalitások

A felhasználók a webes felületen keresztül érhetik el az alkalmazás szolgáltatásait. Biztonsági okokból az alkalmazás különbséget tesz a vendég, valamint a regisztrált felhasználók

között. A továbbiakban bemutatásra kerülnek a két felhasználói kategória által elért funkcionalitások, szerepkörökre lebontva.

1.2.1. Vendég felhasználó

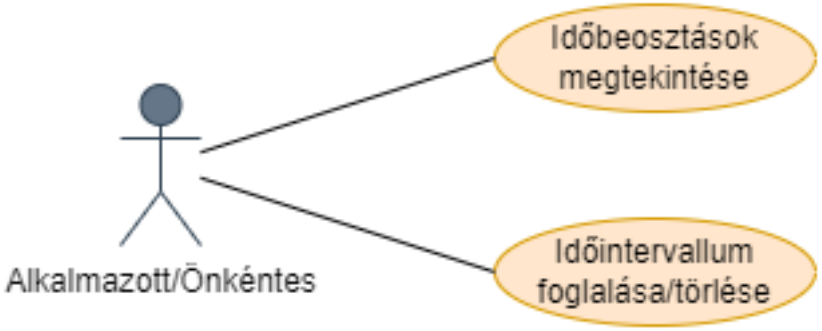
Mivel a hívásátirányítás automatizálásával könnyen vissza lehet élni, és illegális tevékenységekre lehet használni, a vendég felhasználóknak az oldal funkcionalitásai nem elérhetőek. Esetükben egy kezdőlap jelenik meg, ahol a weboldal üzemeltetőinek elérhetőségei vannak feltüntetve.

1.2.2. Regisztrált felhasználó

A rendszer regisztrált felhasználóit három különböző szerepkörbe lehet sorolni: **alkalmazott/önkéntes, szervezet vezető** és **rendszeradminisztrátor**.

Alkalmazott/Önkéntes szerepkör

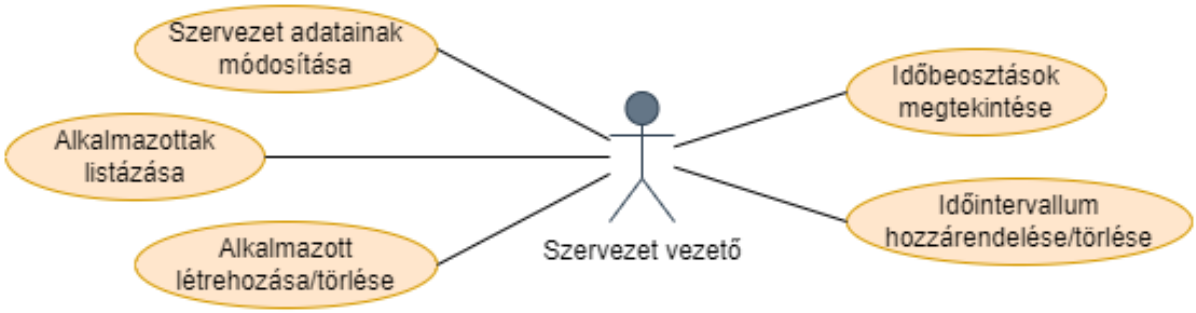
A szerepkör felhasználói az interaktív naptáron keresztül figyelemmel követhetik saját szervezetük időbeosztását, valamint szabad időintervallumokat foglalhatnak, vagy a meglévő foglalásaikat törölhetik.



1. ábra. Az alkalmazott/önkéntes szerepkör funkcionalitásai

Szervezet vezető szerepkör

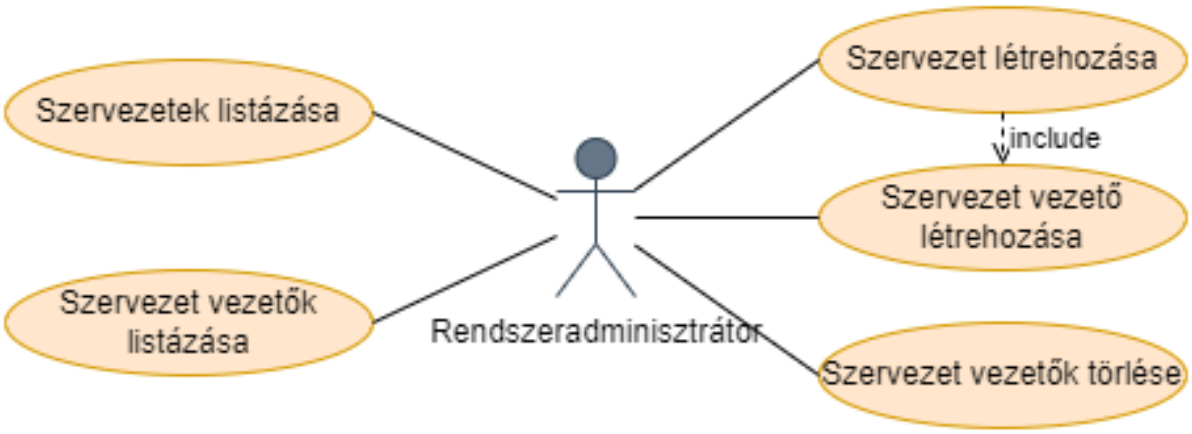
Egy vezető feladatköréhez tartozik a szervezet adatainak kezelése: módosíthatja a szervezet adatait, kijelölhet újabb vezetőket, alkalmazottakat hozhat létre vagy törölhet. Ezek mellett, fontos szerepe van a hívásátirányítás folytonosságának biztosításában: az interaktív naptáron időpontokat rendelhet az alkalmazottaihoz, vagy szerkesztheti az időbeosztást, biztosítva a teljes munkaidő lefedését.



2. ábra. A szervezet vezető szerepkör funkcionalitásai

Rendszeradminisztrátor szerepkör

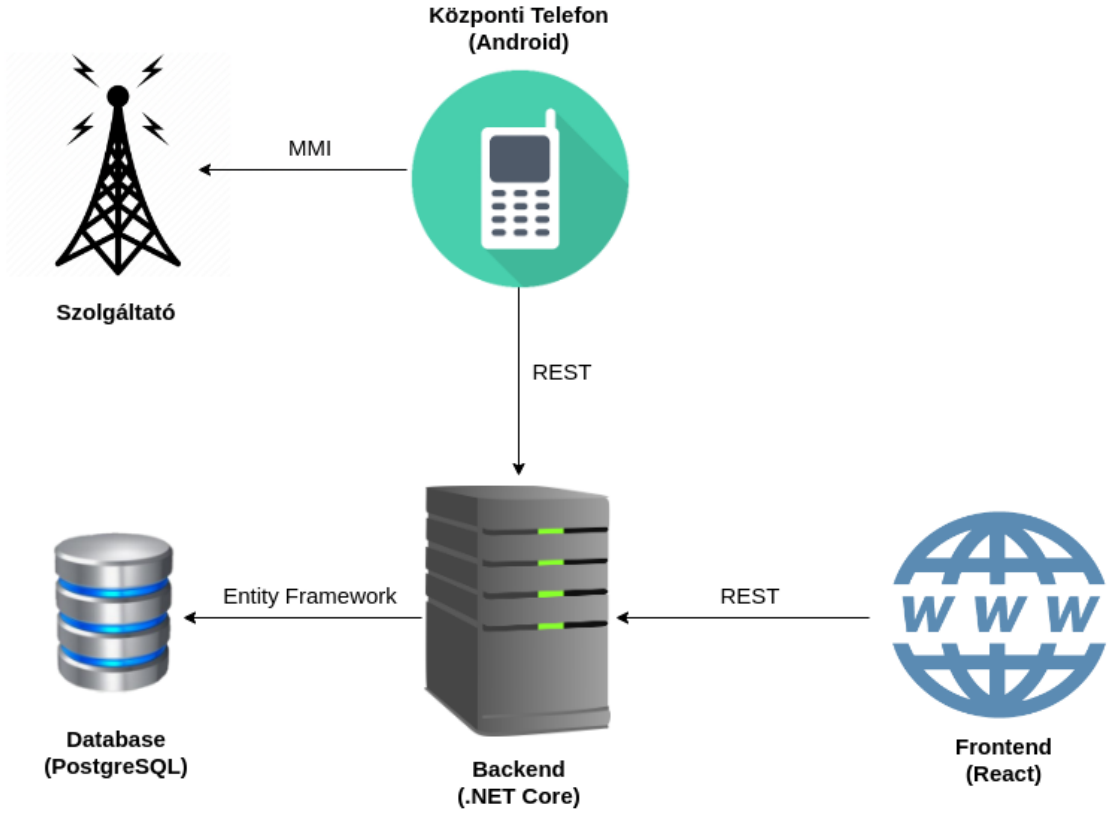
Feladata a rendszer karbantartása. Problémák esetén az adminisztrátoroknak lehetősége van törölni a vezetőket vagy újakat létrehozni. Ezek mellett, a szerepkör tagjai felelnek a rendszer biztonságáért. Ha egy vendég felhasználó igénybe szeretné venni a saját szervezetéhez a Helpline szolgáltatásait, ennek a szerepkörnek a tagjaival kell felvennie a kapcsolatot. Pozitív elbírálás esetén, a rendszeradminisztrátor létrehozza az új szervezetet és a hozzá tartozó vezető fiókját.



3. ábra. A rendszeradminisztrátor szerepkör funkcionalitásai

1.3. Architektúra

Az alkalmazás három jól elkülöníthető részre osztható fel: **backend**, **frontend**, illetve **mobilalkalmazás**. A legalsó szinten a backend-et képviselő szerver található (2. fejezet), egy SQL-alapú adatbázissal együtt. A szerver tartalmazza az üzleti logikát. Feladata REST kérések segítségével kommunikálni a webes felülettel (3. fejezet), ezen keresztül kezelhetik a felhasználók a rendszert, valamint kommunikálni az intézmények központjában megtalálható



4. ábra. A projekt architektúrája, komponensek közti kommunikáció.

mobilapplikációval (4. fejezet), amely MMI kódok segítségével kéréseket küld a mobil szolgáltatóhoz, ahol megtörténik a hívások átirányítása. A komponensek közötti kommunikáció megfigyelhető a 4. ábrán.

A rendszer tervezése a *Model View Controller* [16] minta alapján történt, a szerver oldali vezérlők szolgáltatnak adatokat a webalkalmazás számára, DTO (Data Transfer Object) [13] objektumok formájában. A DTO tervezési minta használata indokolt, mivel az adatok sok esetben nem a modell osztálynak pontosan megfelelő formában érkeznek a szerverhez, valamint sok esetben nem küldhetőek el ugyanabban a formában, például biztonsági okok miatt. A mobil applikáció felelős a beosztás alapján a hívások átirányításáért, amihez a használt architektúrában elengedhetetlen egy SIM kártyával rendelkező eszköz.

2. A szerver

A rendszer központi eleme a szerver, ami az adatok kezeléséért és tárolásáért felelős. A szoftver a *multi-layered architektúra* [28] elveit követve készült, ami kihatással van a projekt teljes szerkezetére. Az alkalmazás több alprojektből épül fel, melyek a program jól elszigetelhető, moduláris komponenseit tartalmazzák, ahogy ez az 5. ábrán látható. Mindegyik layer csak az alatta lévőtől függ, valamint használja a modellosztályokat és a DTO objektumokat tartalmazó Helpline.Common alprojektet, és a Helpline.Contracts modult, amely a rétegek viselkedését meghatározó interface-eket gyűjti egybe.

Az interface-ek moduláris jelleget biztosítanak a rétegeknek, flexibilitást adnak a szoftver számára. Egy réteg teljesen kicserélhető, függetlenül más rétegektől, egészen addig, amíg az adott komponensek megfelelnek az interface-ek által előre definiált viselkedésnek. Ezáltal a rétegek egymásra fekete dobozként képesek tekinteni, teljesen függetlenek a többi réteg implementációs részleteitől.

2.1. Technológia

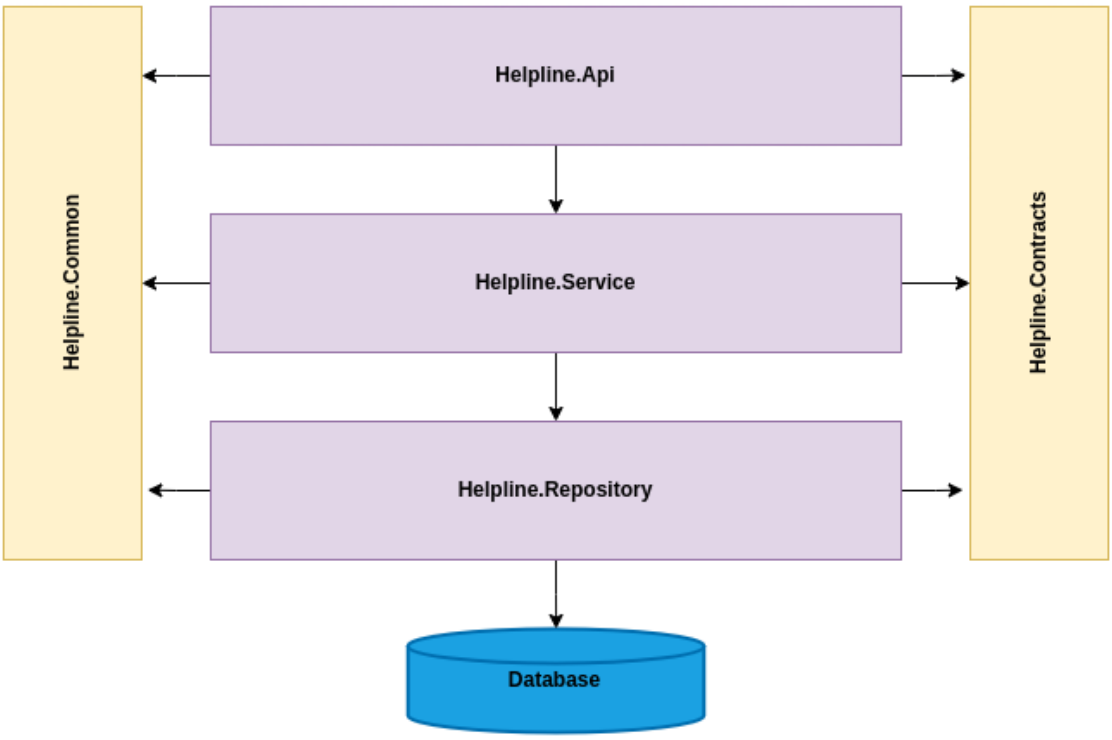
A szerver kódállománya *C#* nyelvben van megírva, *.NET Core* [7] keretrendszer felhasználásával. A keretrendszer megvalósítja az *Inversion of Control* (IoC) [18] elvet, az osztályok függőségei a *dependency injection* [6] minta alapján vannak kezelve. Ez tovább könnyítette a multi-layered architektúra megvalósítását, a komponensek közötti szoros függőségek feloldásával.

Az adatok tárolása *PostgreSQL* [14] adatbázis segítségével történik, amelyet a csapat kombinálva használt a *DBeaver* [8] adatbázis-kezelő rendszerrel.

Továbbá, felhasználásra került az *Entity Framework* keretrendszer is, melyet a 2.2. alfejezet tárgyal részletesebben.

2.2. Adatmodell

A környezeti elemzést követően a csapat implementálta a szükséges modell osztályokat, amelyek a Helpline.Common alprojektbe kerültek. Mivel egyértelmű a kapcsolat a modell osztályok és a relációs adatbázis táblák között, az object-relational impedance mismatch [4] feloldása egyszerűen megtörténhetett. A csapat csak a modellosztályokat írta meg, a *code first approach* megközelítés alapján, melyekből a használt *Entity Framework* [10] ORM



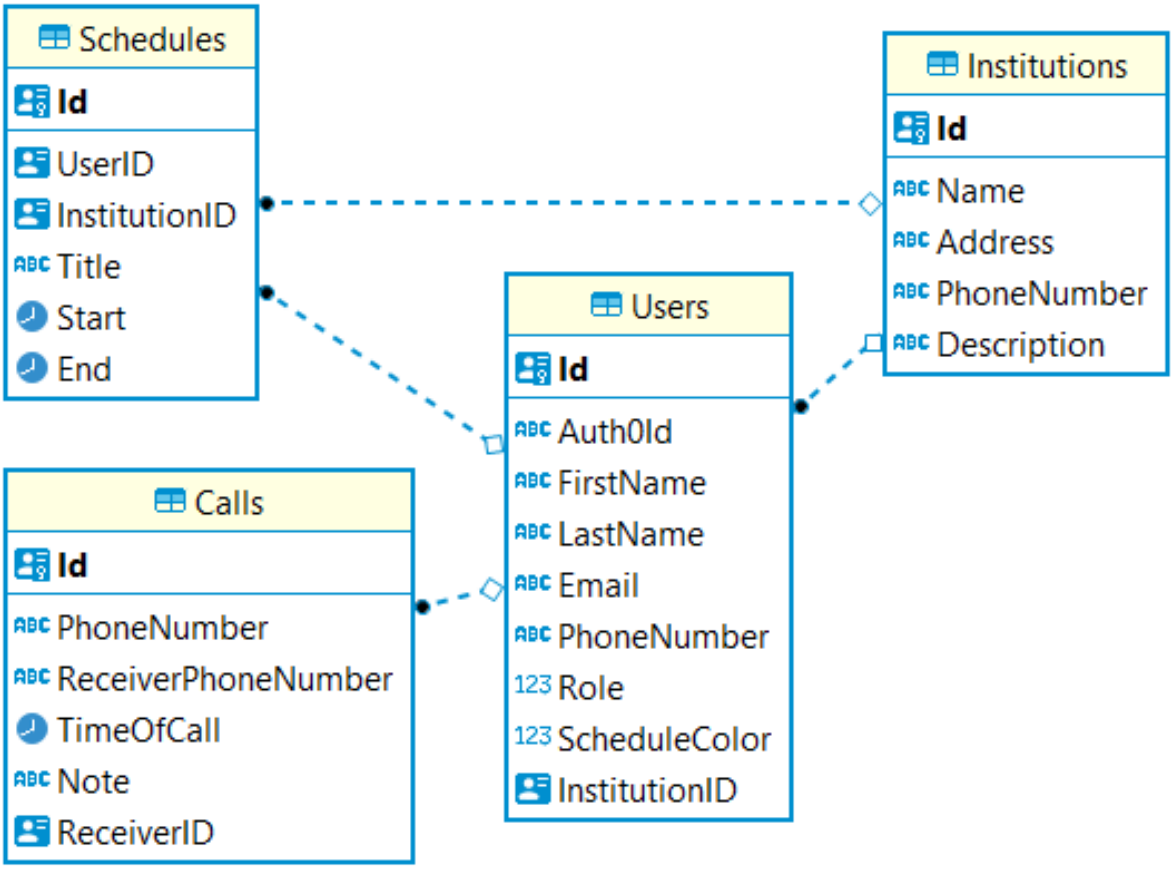
5. ábra. A szerver rétegekre bontva.

keretrendszer generálta a megfelelő adatbázis táblákat a köztük lévő kapcsolatokkal együtt, ahogy ez a 6. ábrán látható.

A keretrendszer, az adatbázis tábláinak generálása mellett, megvalósította a modell osztályok automatikus perzisztálását és adatelérési réteget generált a szerver számára, amelyet a Helpline.Repository alprojektben találhatunk meg. A *RepositoryContext*, amely a keretrendszer által biztosított *DbContext* osztályból származik, tárolja a karbantartandó modellek listáját, lehetőséget biztosít a keretrendszer működésének konfigurálására. A RepositoryBase osztály általános CRUD műveleteket definiál a modellek számára, melyből további specifikus Repository osztályok származnak, ahol a standard műveletektől eltérő adatelérési metódusok találhatóak.

2.3. API

A szerver áll az alkalmazásrendszer központjában, szolgáltatásokat nyújt a webes kliens (3. fejezet) és a mobilapplikáció (4. fejezet) számára. A kommunikáció a REST [17] elvek alapján történik, HTTP protokoll használatával. A Helpline.Api réteghez tartozó controller



6. ábra. Adattáblák és a köztük lévő kapcsolatok.

osztályok feladata egy adott erőforrással kapcsolatos kérések kiszolgálása. A controller osztály a konvencionálisan felépített URL, a HTTP metódus, valamint a query paraméterek alapján routolja a hívást a megfelelő kódrészletet tartalmazó függvényhez. A szerver a kérésekre kizárólag JSON formátumú adattal, illetve megfelelő HTTP státuszkóddal válaszol.

Az 1. kódrészletben látható InstitutionController osztály GetInstitutionById metódusára azok a HTTP GET kérések érkeznek, amelyeket az "api/institution/{id}" URL-re küldtek, ahol az id egy PATH variable. A függvény először az institutionService segítségével lekérdezi a felhasználó által megadott id-vel rendelkező intézményt, majd ezt átalakítja a mapper segítségével a kimeneti adatnak megfelelő DTO objektummá. Ha az URL-be beépített id egy valóban létező intézményt azonosít, akkor az API visszaküldi a kért adatot JSON formátumban a 200-as státuszkóddal együtt. Ellenkező esetben a kérés egy nem létező intézményre vonatkozik, a felhasználó válaszul a 404-es kódot kapja.

2.4. Hitelesítés és engedélyezés

Az 1.2 fejezetben ismertetett szerepkörök az alkalmazás biztonságának alapját szolgálják. A szerepkörökhöz jogosultságok társulnak, jelölve, hogy mely funkcionalitások elérhetőek számukra. A szerverhez érkező kérések végrehajtásához szükséges ellenőrizni, hogy az adott

```
[Route("api/[controller]")]
[ApiController]
public class InstitutionController : ControllerBase
{
    ...

    [HttpGet("{id}", Name = "InstitutionById")]
    [Authorize("SuperAdmin")]
    public IActionResult GetInstitutionById(Guid id)
    {
        var institution = mapper.Map<InstitutionDto>(
            institutionService.GetInstitutionById(id)
        );

        return institution == null ? NotFound() : Ok(institution);
    }

    ...
}
```

1. kódrészlet. Institution lekérdezése id alapján

felhasználó be van jelentkezve, valamint rendelkezik a megfelelő jogosultságokkal a kérés végrehajtásához.

A felhasználók hitelesítése az Auth0 [2] hitelesítési rendszeren keresztül történik. Az Auth0 integrálása széles körben alkalmazott, biztonságos user regisztrációra, bejelentkezésre és elfelejtett jelszófrissítésre nyújt lehetőséget. A rendszer, a biztonság érdekében, az alkalmazás felhasználóinak e-mail címét és jelszavát egy külső adatbázisban tárolja. A bejelentkezéskor az adatbázis alapján elvégzi a felhasználók ellenőrzését. Bár az Auth0 a regisztrációra is biztosít felhasználói felületet, a funkció nem elérhető a vendég felhasználóknak. A regisztráció a rendszer vezetőin keresztül történik, az Auth0 Management API [3] segítségével. Az API lehetőséget biztosít, hogy az alkalmazás szervere új felhasználókat szúrjon be az Auth0 adatbázisába.

A felhasználók az Auth0 saját login felületén keresztül jelentkezhetnek be az alkalmazásba, ahova a 3. fejezetben ismertetett weboldal irányítja át őket. Sikeres bejelentkezés után, a weboldal minden kérés fejlécéhez hozzácsatolja az Auth0-tól kapott *Bearer token-t*, amely alapján a szerver hitelesítheti a felhasználót.

Az Auth0 hitelesítő adatokat tároló adatbázisa mellett, a 2.2. fejezetben ismertetett adatbázisba kerültek elmentésre a felhasználók szerepkörei. A két adatbázis közötti szinkronizálás az Auth0 által biztosított *ID* alapján történik. Ez a folyamat nélkülözhetetlen, mivel ezáltal ellenőrizhető, hogy egy bejelentkezett felhasználó jogosult a kérése végrehajtására. Ha rendelkezik a megfelelő jogosultságokkal, a szerver visszatéríti a kért adatokat, ellenkező esetben 403-as hibakóddal jelzi, hogy a kérés nem teljesíthető.

3. A webalkalmazás

A webalkalmazáson keresztül a felhasználók szerkeszthetik időbeosztásaikat, valamint a szervezetek vezetői is itt láthatják el adminisztrációs feladataikat. A fejezet a webalkalmazás fejlesztéséhez felhasznált technológiákat, módszereket részletezi.

3.1. Technológia

Az alkalmazás webes felületének implementálása *TypeScript* nyelvben, a *React* könyvtár segítségével történt. A TypeScript [29] JavaScript-re épülő, erősen típusos programozási nyelv, ezáltal nagyobb biztonságot nyújt a kód fejlesztésének folyamatában.

A React [21] könyvtár használata lehetőséget nyújtott egy *SPA alapú frontend* elkészítésére. Az SPA (Single-Page Application) egy olyan alkalmazás, amely dinamikusan átírja az aktuális weboldal tartalmát a szervertől kapott új adatokkal, így a böngészőnek nem kell teljesen új oldalakat betölteni, ezáltal az alkalmazás gyorsabbá válik.

A gyorsaság mellett, fontos szempont volt egy felhasználóbarát webalkalmazás elkészítése, ezért a felhasználói felület kinézetét a React könyvtár legnépszerűbb UI keretrendszere, a *Material UI* [20] elemei alkotják.

3.2. Adatok megjelenítése

Az előbb említett SPA alapú alkalmazás megvalósítása React komponenseken keresztül történik. A felhasználói felület oldalai különálló komponensekből épülnek fel, így a webcím változásakor csak a módosult részeket kell lecserélni, frissítve az érintett adatokat.

Az alkalmazás dinamikus útvonalválasztásáért a *react-router-dom* [23] npm-csomag felelős. Lehetőséget nyújt a felhasználók számára, hogy navigálhassanak az oldalak között, karbantartva a komponensek helyes betöltését.

A Helpline webalkalmazás komponensei, amelyek a fontosabb funkcionalitásokat hivatottak megjeleníteni:

- **HomeWrapper**: Az alkalmazás kezdőoldala, mely a felhasználó szerepköre szerint jelenít meg különböző fontosabb információkat.
- **SchedulesPage**: Az alkalmazás interaktív naptárát hivatott megjeleníteni, az alkalmazottak és önkéntesek ezen keresztül jelölhetik be, hogy mikor fogadnának

hívásokat, vagy törölhetik meglévő foglalásaikat.

- ***ListPage, *RegistrationPage**: A szervezetek vezetőinek biztosít felületet az alkalmazottak, önkéntesek karbantartásához.

3.3. Kommunikáció a szerverrel

A kommunikáció REST kéréseken keresztül történik a szerverrel. A kérések feldolgozásáért a Services osztályban található useFetch custom hook felelős. A hook-ok [22] a React 16.8-as verziójával váltak elérhetővé, lehetővé téve az állapot-menedzsmentet és más React funkciók használatát, osztályok írása nélkül.

Ezekre a funkciókra építve az useFetch hook előkészíti a HTTP kérések tartalmát, majd ahogy a 2. kódrészletben is látható, a Fetch API fetch() metódusával aszinkron módon kommunikál a szerverrel. A kérés aktuális helyzete három React állapotban található: az első az aszinkron kérés befejezését jelzi, a második a kapott válaszadatokat menti el, az utolsó a hibaüzenetet tárolja, helytelen működés esetén. Az előző fejezetben megemlített komponensek kéréseket küldhetnek a szervernek a hook-on keresztül, majd a fent említett három állapot szerint frissíthetik saját tartalmukat.

A bejelentkezett felhasználókat a 2.4. fejezetben ismertetett Auth0-tól kapott *Bearer token* hitelesíti a szerveren. Bejelentkezett felhasználó esetén, ennek beállítása a useFetch hook feladata, amely a kérés *Authorization header*-jéhez csatolja az információt. A token az *useAuth0* hook *getAccessTokenSilently* metódusának meghívásával kérhető le. Ha a bejelentkezett felhasználó tokenje időközben lejárt, a hook ezt automatikusan frissíti, így a felhasználónak nem kell újra bejelentkeznie.

```
return fetch(fetchUrl, requestOptions)
  .then((response) => response.json())
  .then((response) => {
        setData(response);
        return response;
  })
  .catch((err) => {
        setError(err.message);
        return err;
  })
  .finally(() => setIsPending(false));
```

2. kódrészlet. Példa a Fetch API fetch() metódusának meghívására és a válasz feldolgozására.

4. A mobilalkalmazás

A mobilalkalmazás egy adott szervezet központi telefonjára kitelepített applikáció, amely az ide beérkező hívásokat irányítja át az időbeosztás szerint aktív alkalmazott telefonjára.

Az Android alkalmazás kódbázisának implementálása Java nyelvben történt. A szerverrel való kommunikációt a Retrofit segítségével valósítja meg.

A Square által kifejlesztett Retrofit [26] egy kliens Android és Java alapú szoftverekhez, amely a Rest API használatát Java interface-eken keresztül teszi lehetővé. A konfigurációhoz szükséges metainformációk annotációk segítségével adhatók meg, ahogy ez a lenti kódrészletben látható. Használatával jelentősen leegyszerűsödik a mobilalkalmazás és a szerver közötti kommunikáció.

```
public interface PhoneCallApi {
    @POST("api/call/")
    Call<PhoneCall> createPhoneCall(@Body PhoneCall phoneCall);
}
```

4.1. Hívásátirányítás MMI kódokkal

Az eredeti terveknek megfelelően a hívásátirányítást az Orange által nyújtott API valósította volna meg, melynek segítségével a backend szerver API hívások használatával konfigurálhatta volna az intézmények központi telefonjaira beérkező hívásokat.

A részletesebb kutatás során kiderült, hogy a szolgáltatást körülményes használni. A legelső megkötés szerint, a központi telefonok kötelesek az Orange szolgáltatót használni, továbbá a használat regisztrációhoz és havidíjhoz kötött. Mivel a költséghatékonyság és a könnyen használhatóság kritikus aspektusai a rendszernek, a csapat elvetette ezt a megoldást.

Az átirányítás megvalósítható *MMI* kódok aktiválásával. A kódok a * vagy # karakterekkel kezdődnek, ezeket tárcsázva különböző információkat kaphatunk a készülékről, ki- illetve bekapcsolhatunk beállításokat. Egy egyszerű példa a GSM telefonokon elérhető *#06# kód, amely megjeleníti a készülékhez tartozó modell és szériaszámot.

A hívásátirányítás (call forwarding) egy szolgáltató által nyújtott funkció, melynek hatására az eredetileg a kliens telefonjára érkező hívások átirányítódnak egy megadott telefonszámra. Ez a szolgáltatás is igénybe vehető MMI kódok használatával. A **61*[telefonszám]*11*05# kód hatására megtörténik a hívások átirányítása a parancsban megadott telefonszámra. Ezután a ##61# kóddal a hívásátirányítás megszüntethető.

Az MMI kódok megfelelő eszközt biztosítanak a hívásátirányítás megvalósítására a mobilapplikáció számára. Könnyen használhatóak, egyszerű tárcsázással aktiválhatóak. Felhasználásuk kevés előfeltételhez kötött, és a hívásátirányítás is ingyenesen elérhető, mivel az alapdíj részét képezi a legtöbb szolgáltatónál. Mivel az átirányítást konkrétan a szolgáltató végzi, az ő kötelessége ennek elérésére kódot biztosítani, valamint annak a formáját eldönteni. Egy hátrányt jelent, hogy nem létezik központi szerv vagy egyezmény, ami kötelezné a szolgáltatókat arra, hogy ilyen funkcionalitást nyújtsanak, illetve a kódok formája nem standardizált. Ez a probléma viszont nem jelentős, mivel a nagyobb európai szolgáltatók többsége biztosítja a szolgáltatást, és egységesen az említett kódot használják.

4.2. Hívásnaplózás

A hívások naplózása egy szükséges alapfunkció sok szervezet számára, és olyan információkat biztosít, amelyekből később különféle statisztikák készíthetőek. Ugyanakkor a hívásátirányításra alkalmazott megoldás miatt a naplózás megvalósítása nem triviális feladat. Egy készülék, amin aktív az átirányítás opció, semmilyen értesítést nem kap a felé intézett hívásokról, hiszen azok már a szolgáltató szintjén át vannak irányítva. Ezért a Helpline rendszer esetében a fentebb említett kód valójában csak 5 másodperc kicsengés után hajtja végre az átirányítást, lehetőséget biztosítva a központi telefonnak, hogy tudomást szerezzen a hívásról, és elvégezhesse annak naplózását.

Az eljárás hátrányaként megemlíthető, hogy a kliens 5 másodperccel tovább kell várakozzon, amíg a hívása célba ér. Sajnos, ez az időtartam nem csökkenthető tovább ezzel a megoldással, mivel a kódok csak 5 másodperc többszörösét engedik meg, mint késleltetés. Egy másik probléma, hogy a központi telefonnak semmi információja nincs a hívásról az átirányítást követően, azaz nem tud sem arról, hogy a hívás végül célbaért-e, sem arról, hogy mennyi időt tartott. A jelenlegi architektúrával nem is megoldható ezeknek az információknak a kinyerése. A probléma orvosolható lenne egy külön telefonos applikációval, amely az alkalmazottak telefonján futna, és képes lenne az említett adatokat továbbítani a szervernek. Egy ilyen applikáció beépítése a rendszerbe a továbbfejlesztési lehetőségek részét képezi, azzal a megjegyzéssel, hogy nem minden esetben lenne alkalmazható, ugyanis néhány szervezet esetében elvárás, hogy az önkéntesek ne legyenek rákötelezve külön alkalmazás telepítésére saját készülékeikre.

5. Fejlesztési folyamat és eszköztár

A fejezet a fejlesztés során felhasznált módszereket és eszközöket mutatja be.

5.1. Szoftverfejlesztés SCRUM segítségével

A fejlesztési folyamat menedzselésére a *Scrum* [15] agilis programozási módszertant alkalmazta a csapat. A programozói tevékenység két hetes intervallumokból, úgynevezett *sprintekből* épült fel.

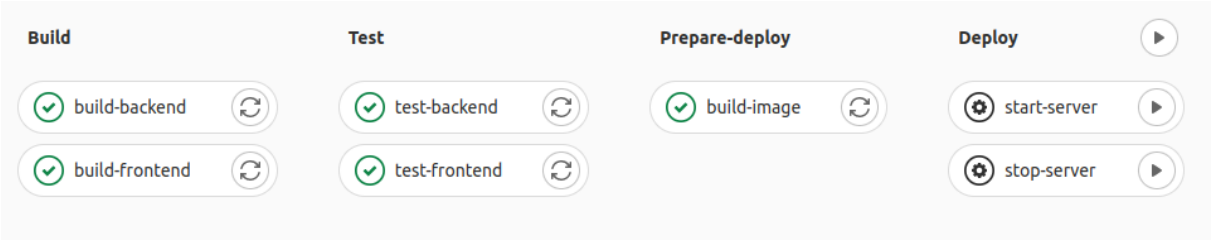
Minden sprint elején a csapat tagjai prioritási sorrendet állítottak fel a megvalósítandó feladatokból, nehézség és hozzáadott érték alapján, majd ezekből beválogattak a lehető legtöbbet, amit az elkövetkező iterációban teljesíteni tudnak. Ezek a feladatok *user story*-k [24] alakjában voltak megfogalmazva. A beválogatott feladatok a Gitlab által biztosított *kanban board*-on [25] átkerültek az összes story-t tartalmazó *open* oszlopból a *todo* oszlopba. Innen a fejlesztők tetszés szerint válogattak feladatokat, melyeket áthelyeztek a *doing* oszlopba, amíg azok fejlesztés alatt álltak. Egy story elkészültekor a feladatot képviselő kártyát átmozgatták az *under review* oszlopa, ahonnan a többi csapattag ellenőrzését követően végül a *merged*-be került. A tábla nagyon hasznos eszköznek bizonyult, gyors, vizuális visszajelzést biztosítva a fejlesztés aktuális állapotáról.

Az iteráció folyamán minden reggel rövid megbeszélést tartottak a csapattagok, amelyen elmondták azt, hogy mivel foglalkoznak, mivel foglalkoztak eddig, valamint az esetleges problémákat amibe beleütköztek. Ez nagyban hozzájárult a fejlesztés gördülékenyebbé válásához.

Minden sprint végén, a megvalósult funkcionalitások bemutatása után, a csapat kiértékelő megbeszélést tartott, elemzésre került a fejlesztés menete az adott iterációban, és a pozitív, illetve negatív élményekből következtetéseket vont le a csapat, a következő sprintek minőségének fejlesztése érdekében.

5.2. Verziókövetés

Verziókövetésre a csapat a Git-et használta, ezen belül a *git flow* [1] elvet. Minden egyes user story saját fejlesztési ágat kapott, ami a develop ágból vált le. Ezen az ágon végbement a fejlesztés, majd a code review-kat követően, befésülődött a develop ágba. A módszer egyik jelentős előnye, hogy a fejlesztési ágak teljesen függetlenek egymástól, valamint hogy a develop



7. ábra. CI/CD pipeline job-ok stage-ekre bontva.

ágon mindig egy stabil, működő verzió található.

A develop ág mellett megtalálható *master* branch feladata a fontosabb mérföldkövek rögzítése, az alkalmazás különböző verzióinak eltárolása.

5.3. CI/CD

Annak érdekében, hogy a develop ág mindig egy működő, helyes, a kódolási konvenciókat megfelelő kódot tároljon, a projekt használta a Gitlab által felkínált CI/CD [12] funkciókat. A központi kódbázis minden módosításakor lefut a 7. ábrán látható *pipeline* [11], ami egymás után végrehajtott *stage*-ek összessége. Minden stádiumban megtalálható egy vagy több úgynevezett *job*, amelyeket egy véletlenszerűen kiválasztott *runner* fog végrehajtani.

Mivel a backendnek és a frontendnek közös Gitlab tárolója van, így a stage-ek jellemzően két job-ot tartalmaznak, melyek közül egyik a frontendhez, másik a backendhez tartozik. Az első stage felelős a komponensek build-eléséért. A következő stage statikus kódellenőrzőket futtat le az alkalmazás adott részein. Ez a frontenden az ESLint [30], a backenden a StyleCop [19] felhasználásával történik. Ezután az automatizált tesztek futtatása következik mindkét alrendszer esetében. Ezek után következnek a kitelepítéshez tartozó stage-k, először a *Prepare-deploy*, ami felépíti az alkalmazáshoz szükséges *docker image*-eket a *Docker Compose* [9] eszköz felhasználásával, illetve feltölti azokat a Gitlab Container Registry-be. Ezt követi egy manuálisan meghívható *Deploy* stage, ahol egy-egy job található a kitelepített alkalmazás elindítására, illetve megállítására.

6. A Helpline működése

6.1. Webes kliens

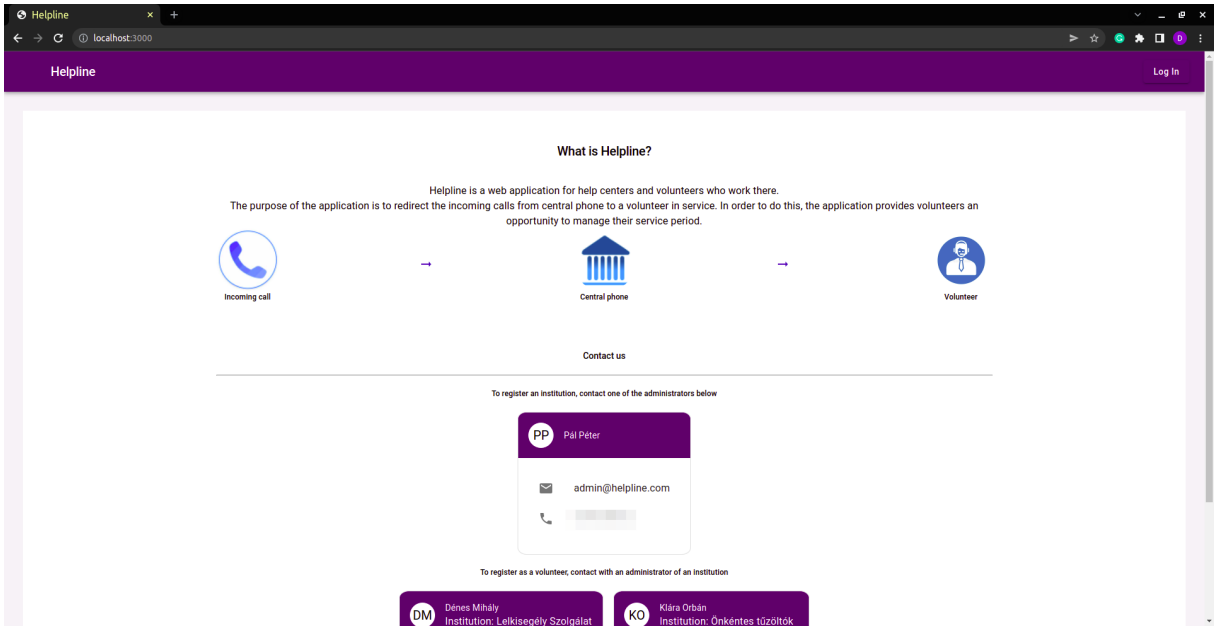
Vendég felhasználó

A webes felületen a be nem jelentkezett (Vendég) felhasználót a 8. ábrán látható oldal fogadja, amely információt tartalmaz az alkalmazás működéséről, valamint elérhetőségi adatokkal szolgál a rendszer adminisztrátorokhoz és szervezet vezetőkhöz. Utóbbi szükségessége indokolt, mivel az alkalmazottak regisztrálása egy adott szervezethez a szervezet vezetőkön keresztül történik.

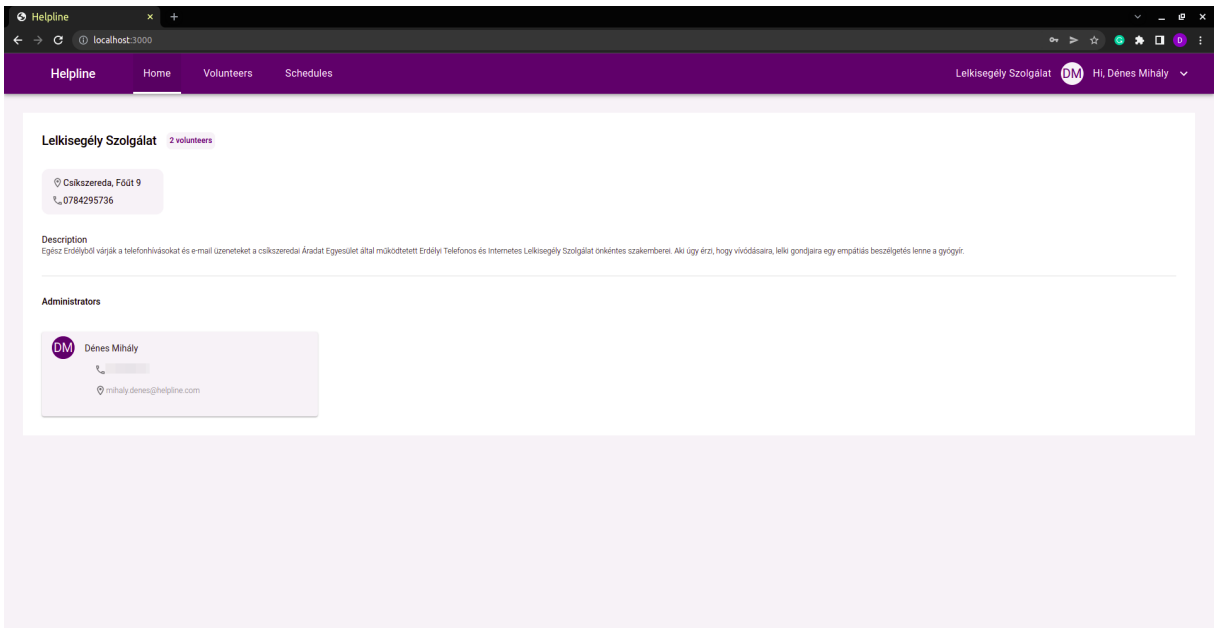
Szervezet vezető

Szervezet vezetőként a bejelentkezett felhasználókat köszöntő, 9. ábrán látható főoldal jelenik meg, amelyik az intézménnyel kapcsolatos adatokat jeleníti meg, például a megadott nevet és leírást, elérhetőséget, az ide tartozó szervezet vezetőket. A felhasználó a *header bar*-ba beépített menün keresztül képes navigálni az oldalak között.

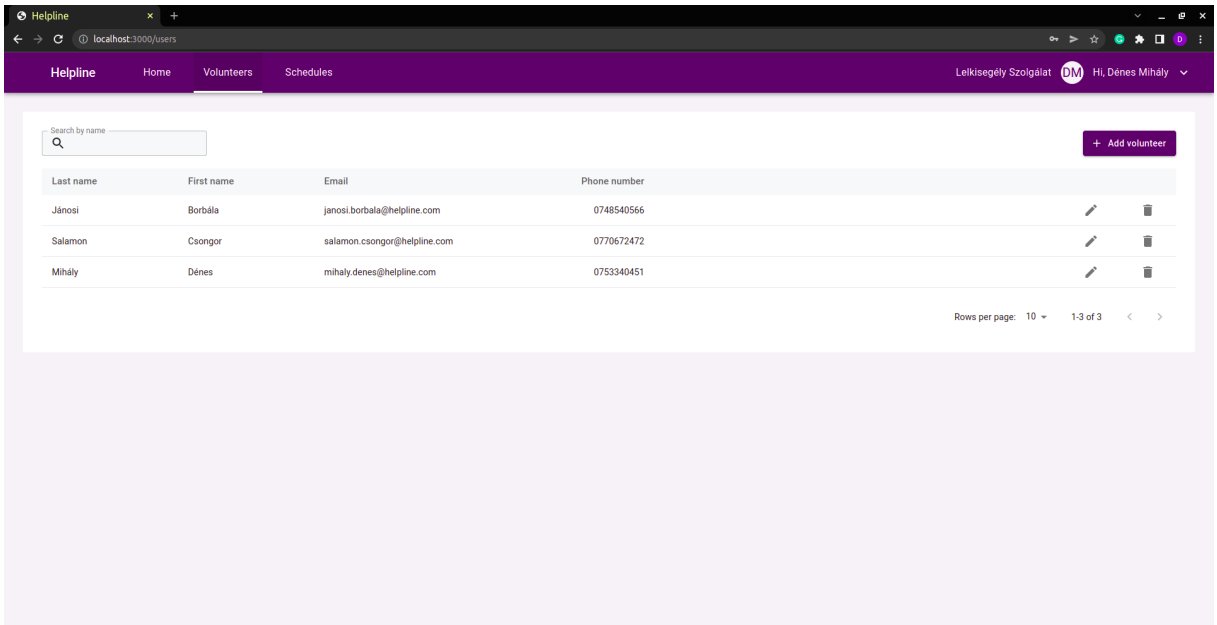
A **Volunteers** menüpont alatt eléri az intézményhez tartozó alkalmazottak/önkéntesek listáját, ahogy ez a 10. ábrán látható. Itt a vezetőnek lehetősége van név szerinti szűrést végrehajtani a felhasználó listára, valamint új felhasználót regisztrálni az intézményéhez az **"+ Add volunteer"** gombra kattintva, mely átirányítja a 11. ábrán látható regisztráció űrlapra.



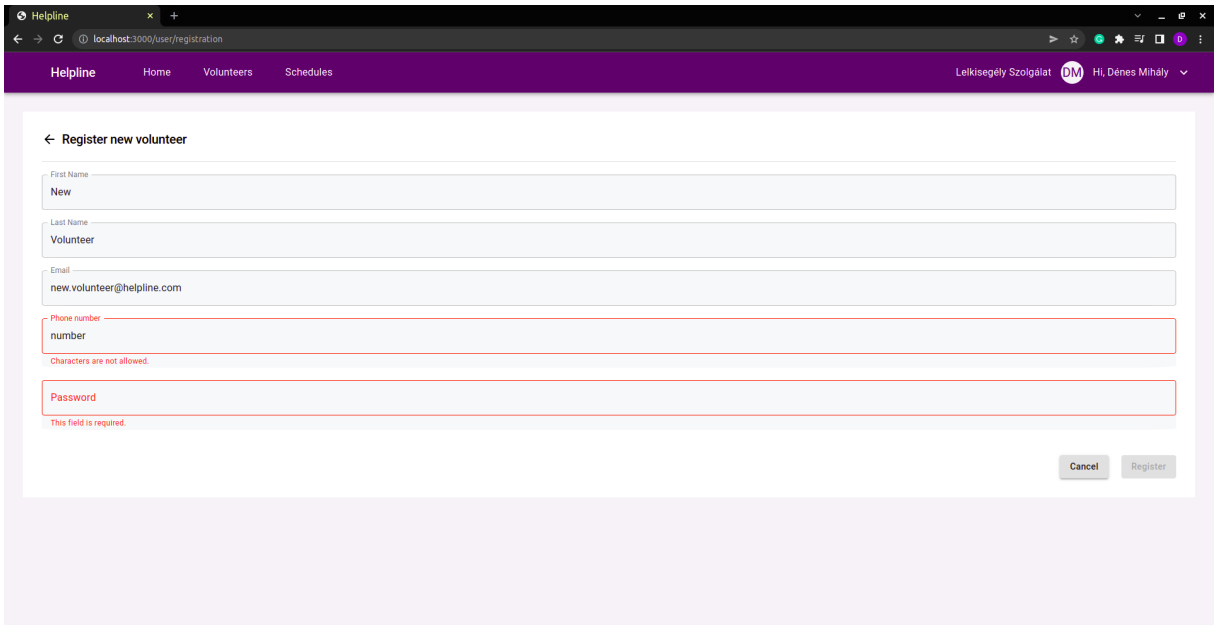
8. ábra. Vendég felhasználót köszöntő oldal.



9. ábra. Szervezet vezető főoldala.



10. ábra. Felhasználó lista.



11. ábra. Alkalmazott regisztrálását lehetővé tevő űrlap.

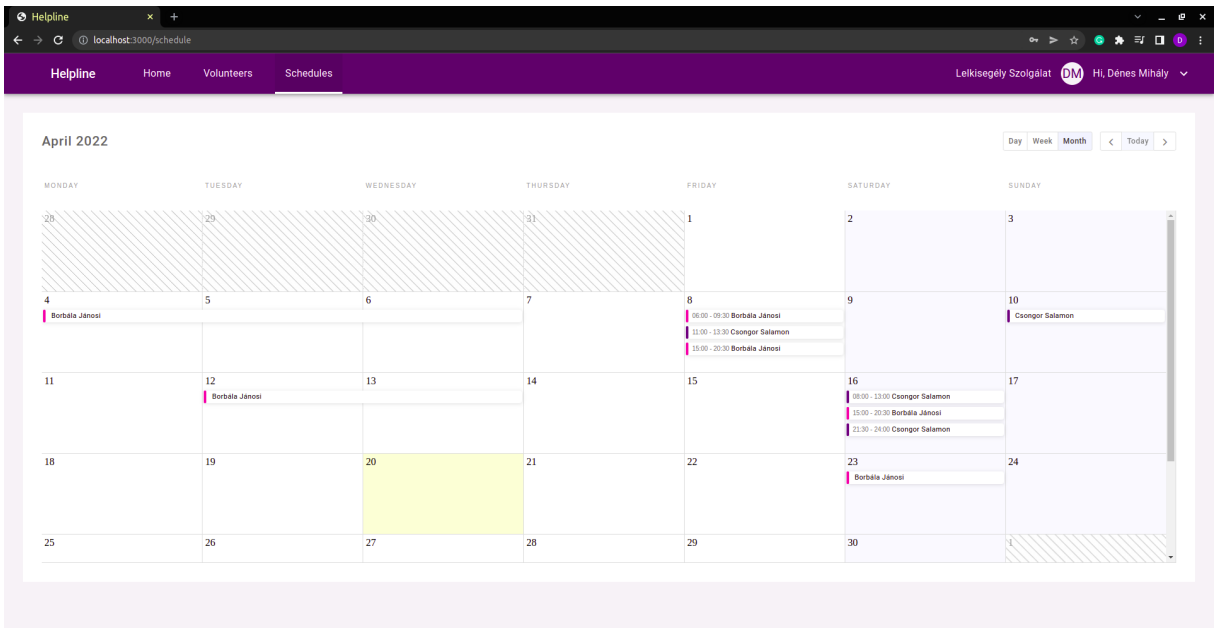
A navigációs menü **Schedules** pontján keresztül eléri az interaktív naptár nézetet (12. ábra), ahol láthatja az intézményhez tartozó alkalmazottak által lefoglalt időintervallumokat, szakaszok formájában. Az alapértelmezett havi nézetből átválthat napi, illetve a 13. ábrán látható heti nézetre. Ebben a nézetben a nem egész napra kiterjedő foglalások is megjelennek.

A naptár nézetben a szervezet vezetőjének lehetősége van időintervallumok lefoglalására egy kiválasztott felhasználó számára, továbbá képes a foglalások törlésére is.

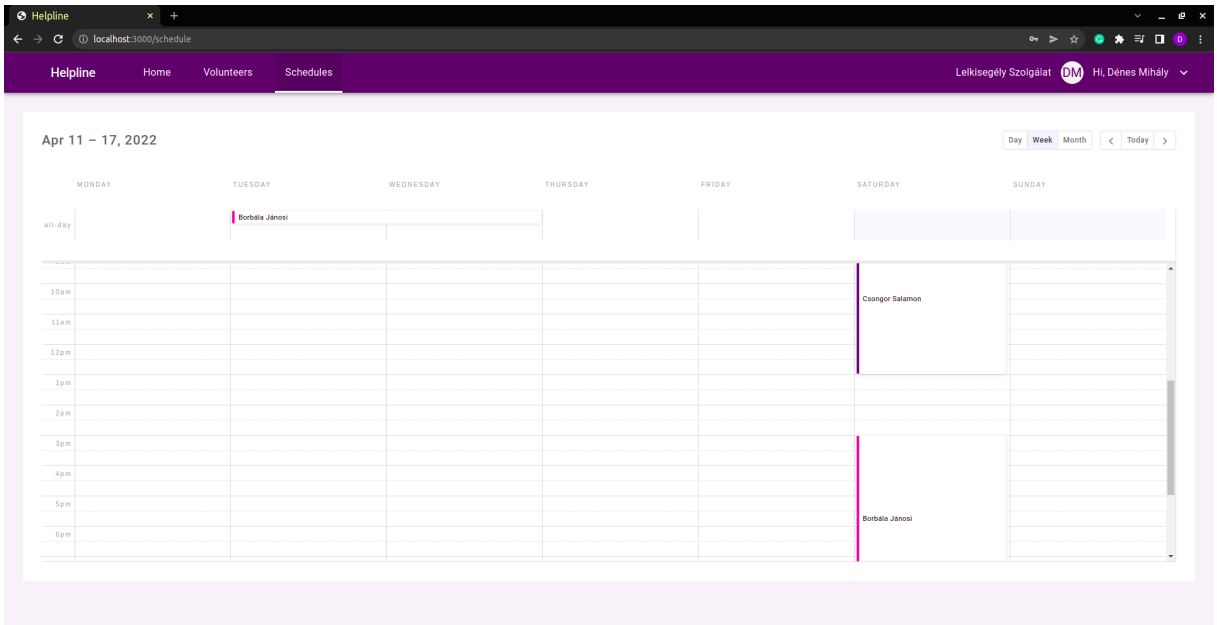
Alkalmazott/Önkéntes

Alkalmazott/Önkéntes szerepkörrel rendelkező felhasználóval bejelentkezve, a naptár oldalon lehetőség van új szolgálati időintervallum lefoglalására. Ez havi nézetben a választott intervallumhoz tartozó napok, míg heti nézetben órák kijelölésével történik. Opcionálisan megadható egy cím is a foglalásnak, ez látható a 14. ábrán. Az alkalmazott és önkéntes szerepkörök gyakorlatilag ugyanazt a felhasználótípust képviselik. A megnevezésbeli különbség csupán a felhasználóhoz tartozó intézmény típusától függ.

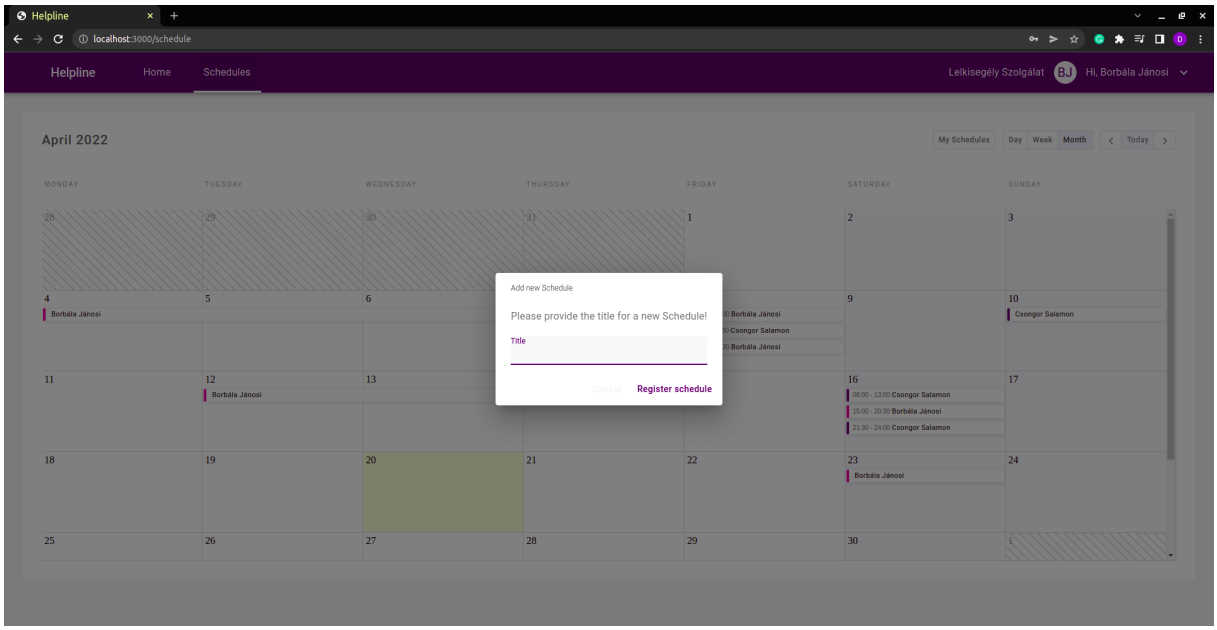
Egy felhasználónak csak az általa lefoglalt időintervallumokat áll jogában törölni. A naptár fejlécében található gomb segítségével lehetősége van a saját foglalásainak megjelenítésére.



12. ábra. Időintervallumok megjelenítése az interaktív naptár havi nézetében.



13. ábra. Időintervallumok megjelenítése az interaktív naptár heti nézetében.

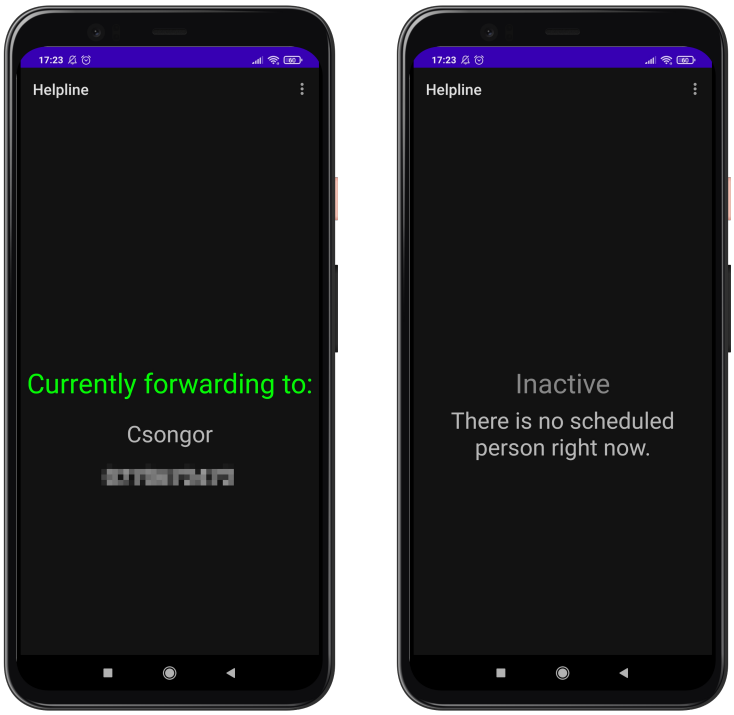


14. ábra. Új foglalás létrehozása.

6.2. Mobilalkalmazás

A szervezetek központi telefonjára kitelepíthető applikáció a szervertől lekérdezi az intézmény beosztását, a beosztás és a jelenlegi idő alapján megvalósítja a hívások megfelelő helyre történő átirányítását. Erről az applikáció visszajelzést is kínál, ami a 15. képcsoporton látható. A jobb oldalon látható állapotot az alkalmazás akkor jeleníti meg, amikor nem talált lefoglalt időintervallumot, melynek része a jelenlegi időpont. Ebben az esetben nincs aktív hívásátirányítás, a központi telefonra beérkező hívások az alkalmazást futtató eszközön csengenek ki.

A bal oldalon látható képen (15. ábra) aktív hívásátirányítást jelez a képernyőn megjelenített alkalmazotthoz, aki a név alatt megjelenített telefonszámmal rendelkezik. A hívásátirányítás aktív az intervallum végéig, vagy amíg egy erre jogosult felhasználó nem törli azt.



15. ábra. A Mobilapplikáció két állapota. Aktív hívásátirányítás (bal), inaktív átirányítás (jobb).

Következtetések és továbbfejlesztési lehetőségek

A fejlesztés során sikerült megvalósítani a projekt fő célkitűzését. Létrejött egy kitelepíthető szoftverrendszer, ami költséghatékony megoldást nyújthat a kisebb szervezeteknek a hívásátirányításaik menedzseléséhez. Emellett, fontos szempont volt egy könnyen használható és egységes, felhasználóbarát alkalmazás elkészítése, melynek létrejöttéhez jelentősen hozzájárultak a webes felülethez elkészített látványtervek.

A megvalósított funkcionalitások mellett számos más továbbfejlesztési lehetőség is felmerült a fejlesztés alatt:

- A központi telefon Android alkalmazása által feldolgozott hívások megjelenítése a webes felületen.
- Hívásátirányítás megvalósítása több egyszerre aktív alkalmazott/önkéntes esetében.
- Mobilapplikáció készítése az alkalmazottak/önkéntesek telefonjára:
  - Egy adott alkalmazott hívásadatainak naplózása.
  - Értesítés küldése a szolgálati idő kezdete előtt.
- Statisztikák készítése a hívások sikerességéről és időtartamáról.
- Minta szerinti időintervallum foglalás.
- Ajánlórendszer bevezetése a szervezet vezetők számára, a szabad időpontok lefedése céljából.

Hivatkozások

[1] Atlassian. *What is Gitflow?* URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (elérés dátuma: 2022. ápr. 13.)

[2] *Auth0 dokumentáció*. URL: https://auth0.com/docs/ (elérés dátuma: 2022. ápr. 22.)

[3] *Auth0 Management API dokumentáció*. URL: https://auth0.com/docs/api/management/v2 (elérés dátuma: 2022. ápr. 22.)

[4] Shashi Bhushan. *Object/Relational Impedance Mismatch.* URL: https://docs.microsoft.com/en-us/ef/ (elérés dátuma: 2022. ápr. 19.)

[5] Dialpad. *Call forwarding*. URL: https://www.dialpad.com/features/call-forwarding/ (elérés dátuma: 2022. ápr. 22.)

[6] .NET hivatalos dokumentáció. *Dependency injection in .NET.* URL: https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection (elérés dátuma: 2022. ápr. 19.)

[7] .NET hivatalos dokumentáció. *What is .NET? Introduction and overview.* URL: https://docs.microsoft.com/en-us/dotnet/core/introduction (elérés dátuma: 2022. ápr. 19.)

[8] DBeaver hivatalos dokumentáció. *DBeaver.* URL: https://dbeaver.com/docs/wiki/ (elérés dátuma: 2022. ápr. 20.)

[9] Docker Compose hivatalos dokumentáció. *Overview of Docker Compose.* URL: https://docs.docker.com/compose/ (elérés dátuma: 2022. ápr. 13.)

[10] Entity Framework hivatalos dokumentáció. *Dependency injection in .NET.* URL: https://docs.microsoft.com/en-us/ef/ (elérés dátuma: 2022. ápr. 19.)

[11] GitLab pipelines hivatalos dokumentáció. *CI/CD pipelines.* URL: https://docs.gitlab.com/ee/ci/pipelines/ (elérés dátuma: 2022. ápr. 13.)

[12] GitLab CI/CD hivatalos dokumentáció. *GitLab CI/CD.* URL: https://docs.gitlab.com/ee/ci/ (elérés dátuma: 2022. ápr. 13.)

[13] Microsoft hivatalos dokumentáció. *Create Data Transfer Objects (DTOs).* URL: https://docs.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5 (elérés dátuma: 2022. ápr. 20.)

[14] PostgreSQL hivatalos dokumentáció. *PostgreSQL.* URL: https://www.postgresql.org/about/ (elérés dátuma: 2022. ápr. 20.)

[15] Scrum hivatalos dokumentáció. *What is Scrum?* URL: https://www.scrum.org/resources (elérés dátuma: 2022. ápr. 13.)

[16] Rafael D. Hernandez. *The Model View Controller Pattern – MVC Architecture and Frameworks Explained.* URL: https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/ (elérés dátuma: 2022. ápr. 20.)

[17] IBM. *REST APIs.* URL: https://www.ibm.com/cloud/learn/rest-apis (elérés dátuma: 2022. ápr. 19.)

[18] Amit Kumar. *Understanding Inversion of Control (IoC) Principle.* URL: https://medium.com/@amitkma/understanding-inversion-of-control-ioc-principle-163b1dc97454 (elérés dátuma: 2022. ápr. 19.)

[19] VisualStudio MarketPlace. *StyleCop.* URL: https://marketplace.visualstudio.com/items?itemName=ChrisDahlberg.StyleCop (elérés dátuma: 2022. ápr. 13.)

[20] *Material UI hivatalos dokumentáció.* URL: https://mui.com/material-ui/getting-started/installation/ (elérés dátuma: 2022. ápr. 12.)

[21] *React hivatalos dokumentáció.* URL: https://reactjs.org/docs/getting-started.html (elérés dátuma: 2022. ápr. 12.)

[22] *React hivatalos dokumentáció. Introducing hooks.* URL: https://reactjs.org/docs/hooks-intro.html (elérés dátuma: 2022. ápr. 12.)

[23] *React-router-dom hivatalos dokumentáció.* URL: https://v5.reactrouter.com/web/guides/quick-start (elérés dátuma: 2022. ápr. 12.)

[24] Max Rehkopf. *User stories with examples and a template.* URL: https://www.atlassian.com/agile/project-management/user-stories (elérés dátuma: 2022. ápr. 13.)

[25] Max Rehkopf. *What is a kanban board?* URL: https://www.atlassian.com/agile/kanban/boards (elérés dátuma: 2022. ápr. 13.)

[26] *Retrofit.* URL: https://square.github.io/retrofit/ (elérés dátuma: 2022. ápr. 7.)

[27] Ringover. *Call forwarding.* URL: https://www.ringover.com/call-forwarding (elérés dátuma: 2022. ápr. 22.)

[28] Packt Editorial Staff. *What is a multi layered software architecture?* URL: https://hub.packtpub.com/what-is-multi-layered-software-architecture/ (elérés dátuma: 2022. ápr. 19.)

[29] *TypeScript hivatalos dokumentáció.* URL: https://www.typescriptlang.org/docs/ (elérés dátuma: 2022. ápr. 12.)

[30] EsLint hivatalos weboldal. *About ESLint.* URL: https://eslint.org/docs/about/ (elérés dátuma: 2022. ápr. 13.)