ETDK-dolgozat

Török István-Dániel

Krizbai Nándor

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

Informatika II.: innovatív számítástechnikai termékek, alkalmazások szekció

Kolozsvár, 2023. május 18–21.

MyOwner

Tárgyak tulajdonosainak megtalálására alkalmas szoftverrendszer

Szerzők:

Török István-Dániel

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

Krizbai Nándor

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

Témavezetők:

dr. Sulyok Csaba, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Nagy-Máthé Orsolya, szoftverfejlesztő,

Codespring

Benedec Botond, szoftverfejlesztő,

Codespring

Kivonat

Az emberek élete során gyakran előfordul, hogy elveszítik személyes tárgyaikat és azokat nehezen vagy szinte soha nem találják meg, így kellemetlen szituációkkal szembesülhetnek.

A MyOwner alkalmazás célja a közösség erejével visszajuttatni az elveszett tárgyakat a tulajdonosukhoz. Az applikációban mindenki felvezetheti saját tárgyainak listáját, majd azokhoz különböző címkéket rendelhet: QR-kód, RFID, illetve egyszerű szöveges kód. Minden tárgyhoz részletes leírást és képeket lehet hozzáadni annak érdekében, hogy a megtaláló minél könnyeben tudja azonosítani azokat. Továbbá a felhasználó különböző elérhetőségeket adhat meg, ahol a tárgyainak megtalálói kapcsolatba léphetnek vele. Amennyiben valaki az alkalmazáson belül olvas be egy címkét, a tulajdonos elérhetőségeit, illetve a tárgy információt fogja látni, valamint lehetősége lesz jelezni a tulajdonosnak, hogy a tárgyát megtalálták, ebben az esetben a tulajdonos push értesítést fog kapni.

A webes felület lehetőséget biztosít a címkék beolvasására azoknak is, akik nem telepítették az alkalmazást a mobil eszközükre, ez esetben a weboldalon láthatják a tulajdonos információit.

A továbbiakban az említett funckionalitásokról, az alkalmazás architektúrájáról és a felhasznált technológiákról láthatunk egy részletesebb bemutatást.

Tartalomjegyzék

Bevezető

Bizonyára mindenki számára ismerős érzés elveszíteni egy fontos tárgyat, legyen az pénztárca, kulcstartó vagy akár egy egész hátizsák. A legnagyobb probléma az, hogy ha egy tárgyat megtalál valaki, akkor nagyon nehezen tudja elérni annak jogos tulajdonosát. Egy példa egy egyszerű kabát, amit valaki a buszon felejt, amit ha valaki megtalál, ha szeretné sem tudja visszaadni a tulajdonosnak. Napjainkban ezen számos csoport próbál segíteni, amiket közösségi oldalakon hoznak létre, viszont ezek korlátozottak abból a szempontból, hogy azok, akik nem tagok vagy éppenséggel nem is használják a felületet, ahol a csoport van, nem fogják látni a hirdetést. Továbbá, a Play Store-on található *Get it Back - Lost & Found*[1] alkalmazás hasonlóan az elveszett tárgyak problémáját próbálja megoldani. Az egyedüli hátránya az említett applikációnak, hogy nehéz felvenni a kapcsolatot a tulajdonossal, mivel számos bejegyzést kell végig böngészni, amíg a megfelelő személlyel fel tudjuk venni a kapcsolatot.

A MyOwner applikáció könnyíteni próbál a kapcsolatfelvétel folyamatán tulajdonos és megtaláló között. Az applikációban regisztráció után mindenki listát vezethet a tárgyairól, amikhez különböző információkat is csatolhat: egy leírást és akár képet is. Ugyanekkor, tárgyakat elveszettnek nyilváníthat. Mindenki lehetőséget kap a saját elérhetőségi listája kezelésére, amiben e-mail címet, telefonszámot vagy lakcímet tárolhat el.

Az applikáció fő funkcionalitása a tárgyakhoz rendelt címkék létrehozása és azok beolvasása. Ezekből három típust támogat az alkalmazás: QR kód, RFID és szöveges kód. A QR kódot számos módszer segítségével lehet exportálni az applikációból, majd ezt kinyomtatva a tárgyra ragasztani. A szöveges kódot akár kézzel írott vagy nyomtatott szöveg formájában lehet a tárgy mellé csatolni. Az RFID számára az applikációban található egy modul, amely segítségével az RFID-hoz társítja egy tárgy ID-ját, majd leolvasáskor megkeresi az RFID-hoz társított tárgyat és az ahoz tartozó információkat jeleníti meg. Megtaláláskor a felhasználó ezeket beolvashatja az applikáció vagy weboldal segítségével.

A dolgozat hátralevő részének a struktúrája a következő: az 1. fejezetben a szoftverrendszer funkcionalitásairól van szó, a 2. fejezetben a rendszer architektúrájáról, illetve a szerver és kliens felépítéséről és megvalósításáról kerül szó. A 3. fejezetben a fontosabb technológiákról és eszközökről található egy-egy rövidebb leírás. A 4. fejezetben a munka folyamata

[1] https://play.google.com/store/apps/details?id=com.ims.getitback.lostandfound&hl=en&gl=US

kerül bemutatásra. Az 5. fejezetben egy részletesebb betekintést nyerhetünk az applikáció mükődésébe, míg a 6. fejezetben továbbfejlesztési lehetőségekről és következtetésekről esik szó.

A projekt fejlesztése 2022 júliusban kezdődött, a Codespring Mentorprogram[2] keretén belül. A projekt nyári gyakorlat keretében indult el, majd folytatódott 2022 őszén kezdődő félévben, a Csoportos Projekt tantárgy részeként. Szeretnénk megköszönni dr. Sulyok Csabának a koordinálást, Nagy-Máthé Orsolyának és Benedec Botondnak a szakmai eligazítást és mentorálást. Köszönet illeti továbbá, a tantárgy keretein belül csatlakozott diáktarsainkat, Opra Júliát, Pulbere Davidot, Székely Róbertet és Török Szabolcsot a projekthez való hozzájárulásért.

[2] https://edu.codespring.ro

1. Funckionalitások

Az alkalmazás a hatékony használat érdekében különbséget tesz vendég, illetve regisztrált felhasználók között. A továbbiakban bemutatásra kerülnek a funkcionalitások, melyek a szerepköröknek megfelelően lesznek csoportosítva.

1.1. Vendég felhasználó

Egy megtalált tárgy kódjának leolvasása nem igényel bejelentkezést, ezzel megkönnyítve a kapcsolatfelvételt a fiók nélküli felhasználóknak. A vendég felhasználónak továbbá lehetősége van jelezni, hogy valakinek megtalálta a tárgyát. A tulajdonos számára egy értesítés fog érkezni minden olyan eszközre, amin a tulajdonos felhasználója van bejelentkezve.

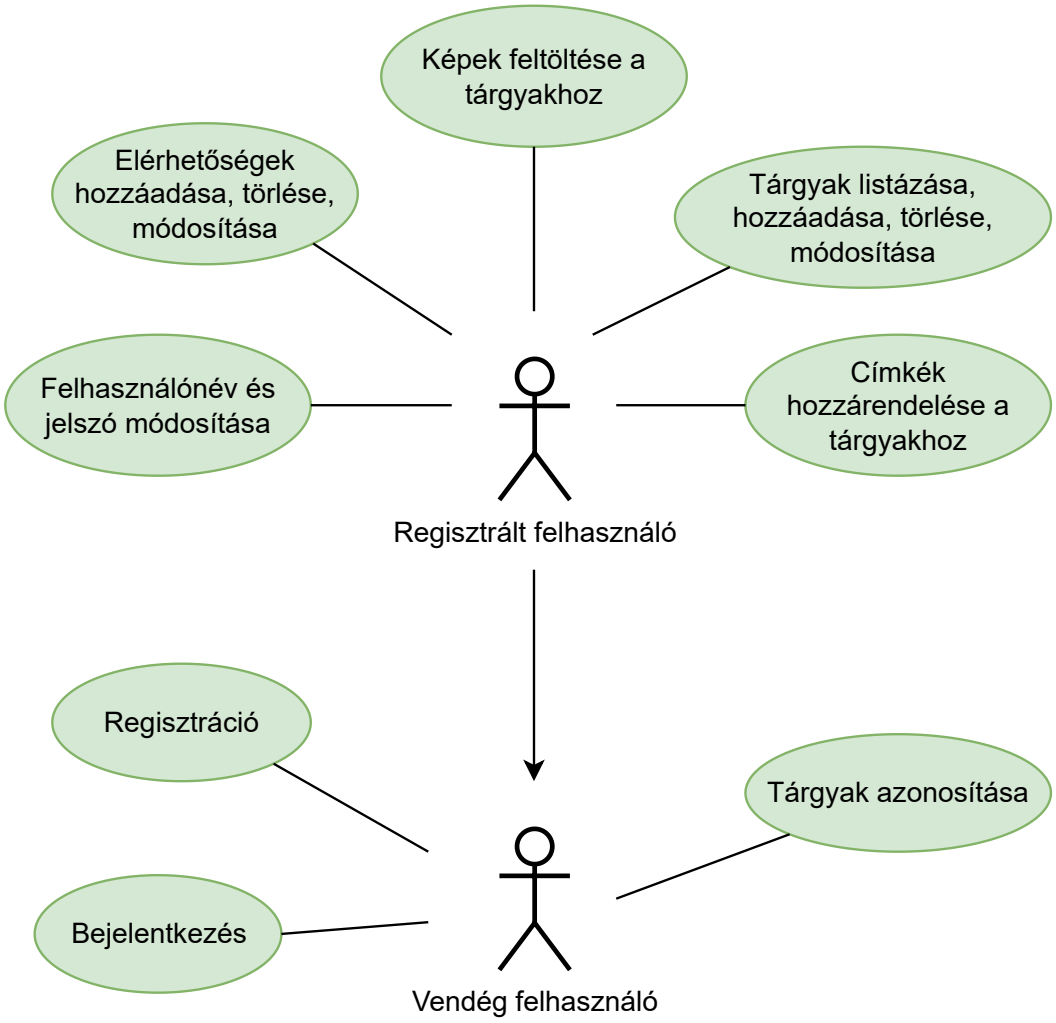
Regisztráció során a felhasználónak meg kell adnia a nevét, e-mail címét, felhasználó nevét és jelszavát. Biztonsági okokból a jelszónak kötelezően tartalmaznia kell nagy betűt, számot és speciális karaktert. Sikeres bejelentkezés esetén az alkalmazás bezárása után nem szükséges még egyszer bejelentkezni, illetve kijelentkezéskor a felhasználó a kezdőoldalra lesz átirányítva.

1.2. Regisztrált felhasználó

A regisztrált felhasználók a vendégek jogai mellett további funkcionalitásokkal rendelkeznek (lásd az 1. ábrát). Minden felhasználó felvezethet bármennyi tárgyat, majd ezek információit tetszés szerint módosíthatja. Minden tárgyhoz opcionálisan meg lehet adni képet és leírást is, annak érdekében, hogy a megtaláló a lehető legkönnyebben biztosra tudjon menni, hogy valóban azt a tárgyat találta meg, amelyhez a címke tartozik. Továbbá a listában ki lehet választani, mely tárgyak élvezzenek prioritást a megjelenítéskor, így a fontosabb tárgyakat a lehető legkönnyeben nyomom tudja követni a felhasználó. Meg lehet adni az elérhetőségeket, amiken keresztül az elveszett tárgy megtalálója felveheti a kapcsolatot a jogos tulajdonossal. Az elérhetőségek csoportosítva vannak telefonszám, e-mail cím és lakcím szerint, illetve bármikor módosítható a láthatóságuk.

A tárgyakhoz háromféle címkét lehet rendelni, amelyek a tárgyak azonosításában játszanak szerepet.

- QR kód - Minden felvezetett tárgyhoz automatikusan generálodik, a tárgy információival egy oldalon találhatjuk meg. A QR kódra kattintva, számos lehetőség adódik annak



1. ábra. Funkcionalitások

exportálására különböző közösségi oldalakon, e-mailen keresztül vagy galériába lementve. Ezek után könnyedén kinyomtatható a kód és fel lehet ragasztani a tárgyra.

- RFID kód - A működéshez a készüléknek rendelkeznie kell NFC technológiával. és egy gombra kattintva a készülékhez közelített RFID-hoz társítja a tárgyat. Majd az applikációba beépített RFID olvasóval a tárgyhoz tartozó információk lesznek megjelenítve.

- Szöveges kód - Létrehozáskor a felhasználó meg kell adjon egy általa választott szöveget. Adatbányászat elkerülése érdekében az applikáció egy véletlenszerűen generált szöveget fűz hozzá a meglévőhöz. Ezután, a tárgy mellé lehet fűzni a szöveget vagy akár kézzel rá is írni. A megtalálónak nincs más dolga mint az applikációban vagy weboldalon a keresőbe beírni a megfelelő szöveget és a tárgyhoz tartozó információkhoz lesz irányítva.

2. Az alkalmazás felépítése

A MyOwner alkalmazás három fő részből áll. Az első rész egy webszerverből, amely kapcsolatban van az adatbázissal és abba menti az adatokat. Továbbá két kliens oldali alkalmazásból tevődik össze, ami egy telefonos applikáció és egy web-kliens. A frontend egy grafikus felhasználói felületet biztosít, melynek segítségével az adatok megjeleníthetőek a felhasználók számára. A backend és frontend rész is többrétegű architektúrával lett megvalósítva. Minden rétegnek más-más feladata van, melyek eredményeit megosszák az egymással való kommunikáció során. A réteges felépítés lehetővé teszi a hatékony és karbantartható fejlesztést, mivel minden réteg a saját felelősségi területét kezeli.

Az alkalmazás struktúrája miatt a kliensoldalról érkező kérések két csoportra oszlanak. Azon kérések amelyek útvonala */api/*-val kezdődik a API-ra lesznek átirányítva, ahol a szerver feldolgozza a kérést, elvégzi a kért műveletet a PostgreSQL adatbázison illetve választ küld *JSON*[3] formátumban. Azon kérések melyek nem a fentebb említett kategóriába esnek, a weboldal szerveréhez lesznek irányítva, ahol az útvonalnak megfelelő statikus oldalakat fogja válaszként kapni. Ennek megvalósításáról a 4.4. fejezetben nyerhetünk részletesebb betekintést.

2.1. Az applikáció

Mivel a telefonos applikáció és a weboldal architektúrailag nagyon hasonló, a továbbiakban csak az alkalmazás felépítéséről nyújtunk részletes betekintést.

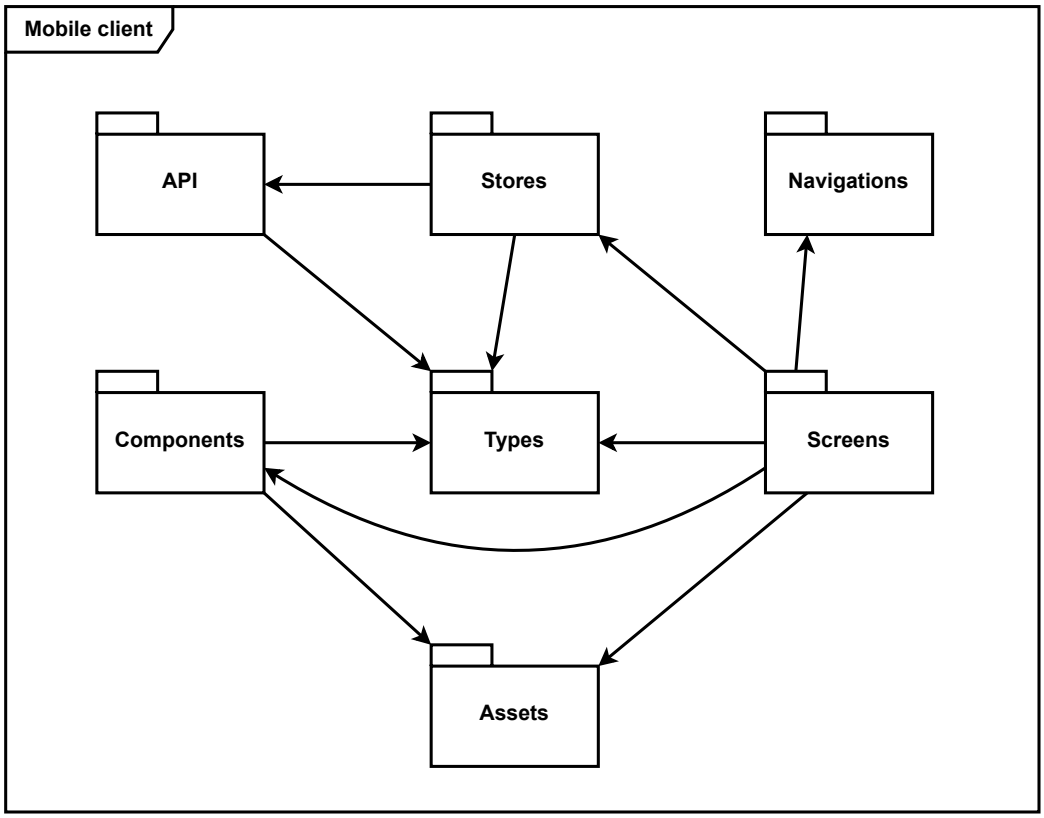
2.1.1. Architektúra

A telefonos alkalmazás hét komponensből tevődik össze (lásd a 2. ábrát).

A *Components* olyan React komponenseket tartalmaz, amelyek újrahasználhatósággal bírnak. Minden komponens önálló funkcionalitással rendelkezik és ezekből épül fel a felhasználói felület. A felhasználói felület kinézete mellett adatokat kezelnek, valamint az alkalmazás működését határozzák meg. A kisebb komponensekre való felosztás könnyebb karbantarthatóságót és hibakezelést eredményez.

A *Screens* képernyőkomponenseket tartalmaz, amelyeket navigációs célra alkalmasak. Minden képernyőkomponens egy sajátos funkciót szolgál ki, például egy regisztrációs képernyőt vagy egy személy adatait bemutató képernyőt. A *Screens a Components* modul

[3] JavaScript Object Notation



2. ábra. A telefonos alkalmazás felépítése

elemeit jeleníti meg a nekik megfelelő oldalon.

A *Navigation* az alkalmazás képernyői közötti navigációt valósítja meg, valamint egy navigációs sávot és a különálló oldalak neveit tartalmazza. A navigációs sáv különböző oldalakra irányít, valamint a különálló oldalakra levő navigációt is kezeli.

A *Types* modul a kliens oldali entitások szerkezeteit definiálja, mint típusokat. Emellett olyan típusok is jelen vannak, amelyek a szerver oldali DTO-k megfeleltetései.

Az *API* modul HTTP kérések küldéséért és a válaszok fogadásáért felel, így lehetővé válik a szerverrel való kommunikáció. Aszinkron kérések által elérhetőek lesznek az adatok, melyeket a szerver küld válaszként. Ezáltal az oldalak dinamikussá válnak, mert mindig a legfrissebb adatok vannak szolgáltatva.

A *Stores* réteg az alkalmazás állapotának frissítését és kezelését segíti elő. Az entitásokat lekérő, módosító és törlő műveleteket gyűjti össze, melyet az összes komponens számára elérhetővé tesz, ezáltal az adatok állapota különálló egységet alkot.

Az *Assets* a különféle statikus fájlokat tartalmazza, melyeket az alkalmazás használ. Ezen könyvtár tartalmát csak importálnunk kell, megadva az szükséges erőforrás útvonalát. Szöveg típusokat, képeket és animációkat tartalmaz.

A webes alkalmazás struktúrája hasonló a telefonos applikáció struktúrájához. Annyiban tér el, hogy hiányzik az *Assets* és a *Navigations* modul. Reactban a navigáció sokkal egyszerűbben megoldható, ezért nem is volt szükség annak elválasztására.

2.1.2. Komponensek

Két fajta React komponens létezik: stateful és stateless. A stateful komponensek állapottal rendelkeznek, amelyet frissítenek. Egy úgynevezett state objektumban tárolják állapotukat, amelyet csak maguk a komponensek módosíthatnak. Az állapot változása maga után vonja a komponens újrarajzolását, amely lehetővé teszi a dinamikus adatok megjelenítését. A stateless komponensek nem rendelkeznek állapottal, csak adatokat kapnak paramétereken keresztül, melyeket props-nak nevezünk. Emiatt az újrarajzolás gyorsabban történhet.

Az előző alfejezetben említett *Components* modulban találhatóak azon komponensek, amelyek a nekik szabott funkciójuk szerint csoportosítva vannak:

- *Indicators*: animációkat jelenítenek meg, amikor a szervertől kell adat érkezzen. Abban a pillanatban, amikor megérkezett az adat, az animáció eltűnik és megjelenik az oldal. Ez nagy szerepet játszik a felhasználói élményben és jelzi a felhasználónak, hogy várnia kell.

- *Inputs*: olyan beviteli komponenseket tartalmaz, amelyek lehetővé teszik a szöveg írását a felhasználók számára és csak kinézetben különböznek. A sorok száma testre szabható, valamint egyikük támogatja az angol helyesírás ellenőrzőt is. Regisztráció, bejelentkezés, illetve új adatok bevitelekor használjuk.

- *Modals*: felhasználói interakció után egy átmeneti ablakot jelenítenek meg. Hibák vagy egy feladat jóváhagyása esetén jelennek meg. A háttérben levő felület nem érhető el, amíg az ablak előtérben van. Ez segít a felhasználónak a megadott feladatra fokuszálnia.

- *Scanners*: a készülék kameráját nyitja meg, miután engedélyt kap hozzá. QR kód leolvasásra és dekódolására alkalmas. Egy tárgy QR kódjának sikeres leolvasása után megjeleníti az ahhoz tartozó információkat.

- *Sliders*: egy sor kép megjelenítését valósítja meg, melynek mérete és animációja módosítható. Megengedi a képek gyors megtekintését és törlését is.

- *Views*: olyan komponenseket tartalmaz, amelyek csak egy bizonyos képernyőhöz tartoznak. Ha egy képernyő három listát jelenít meg különböző adattal, akkor a listát megjelenítő komponens ebben a könyvtárban található.

2.1.3. Kommunikáció

Az applikáció REST kéréseken keresztül kommunikál a szerverrel. A REST (Representational State Transfer) egy architektúrális stílus, melyet az alkalmazások közötti kommunikációhoz használnak [10]. Ezáltal valósul meg a szerverrel való adatcsere, amely lehetővé teszi az adatok dinamikus változását az alkalmazásban. A kommunikáció nélkül nem lenne lehetséges az adatok szinkron frissítése a szerverrel. A HTTP kérés típusa határozza meg azt, hogy az alkalmazás milyen típusú információt küld vagy fogad. A szerver és a kliens is JSON formátumban küldik az adatokat, melyek a megfelelő típusú objektummá lesznek konvertálva. A perzisztencia megőrzése érdekében az újonnan létrehozott vagy módosított adatokat a kliens elküldi a szervernek.

2.2. A szerver

2.2.1. Architektúra

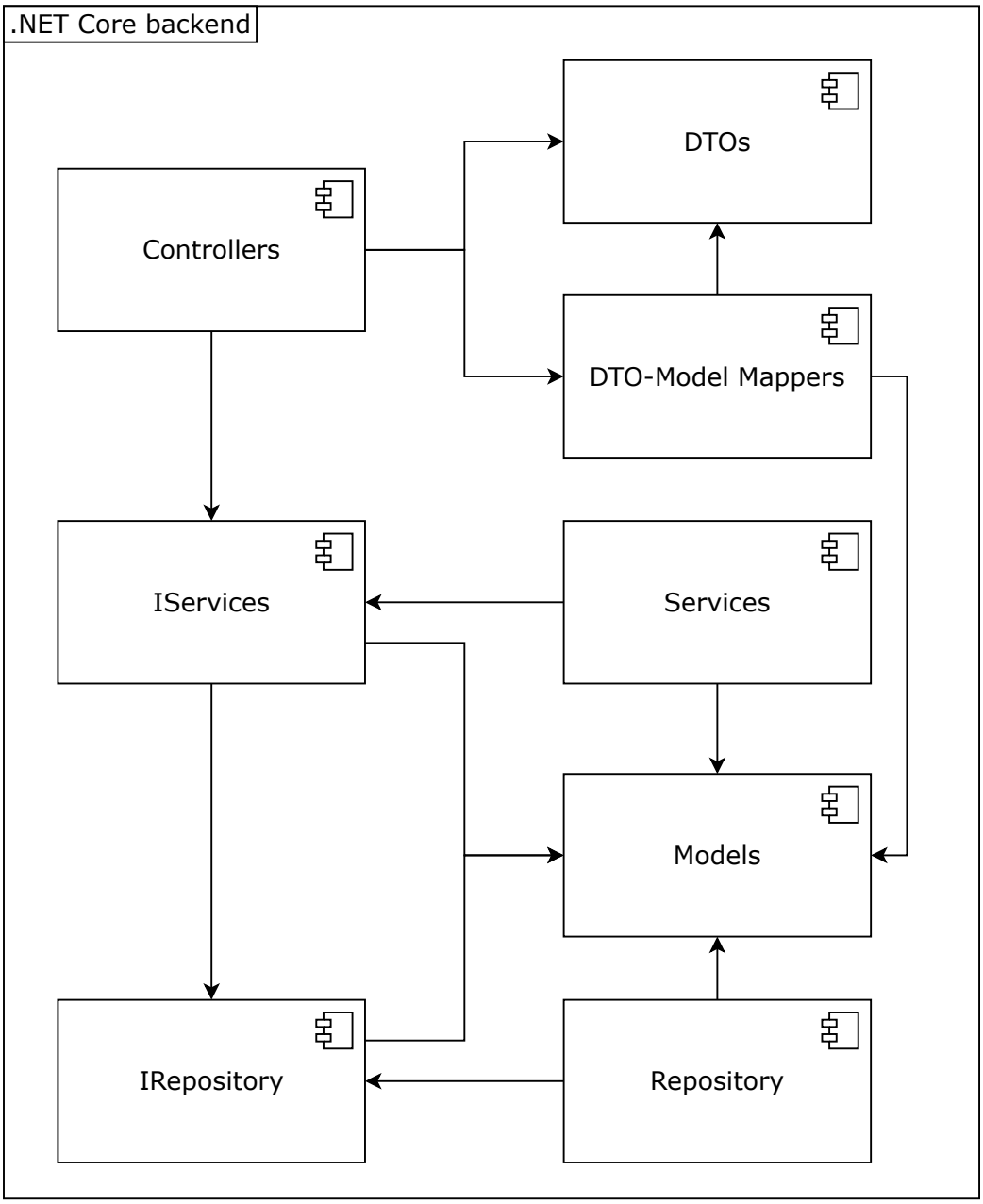
A szerver jól elkülöníthető komponensekre van osztva (lásd a 3. ábrát).

A *Models* tartalmazza azon struktúrákat, amelyekkel a szerver dolgozik és amelyek meghatározzák az adatbázis felépítését is. A *DTOs* a fentebb említett adatstrúkturák sajátos formája, amelyek a kliensekkel történő kommunikációra használandóak.

A *DTO-Model Mappers* feladata a beérkező DTO-kat modelekké alakítani, illetve a kimenő modelleket DTO-kká alakítani. Mindezt a *Controllers* kérésére végzi, aki fogadja a kliens kéréseit és válaszokat küld vissza.

Az *IServices* tartalmazza a *Services* réteghez tartozó interface-eket. A *Controllers*-től érkező kérések ide érkeznek, majd a megfelelő implementációhoz vannak irányítva. Ez a réteg tartalmazza az üzleti logikát, azon műveleteket melyek előkészítik a választ, amit a *Controllers* elküld a kliensnek

Az *IRepository* tartalmazza a *Repository* réteghez tartozó interface-eket. Ezen a rétegen történik a kommunikáció az adatbázissal *ORM keretrendszer* segítségével, itt találhatóak azon beállítások is, amelyek felelnek azért, hogy az adatbázis mindig a *Models*-ben található



3. ábra. A központi szerver architektúrája

adatstruktúrákat reflektálja. A *Services*-től kapott kérések alapján a megfelelő *CRUD* műveletet (létrehozás, olvasás, frissítés vagy törlés) végzi el.

Az *IRepository* és *IServices* segítségével a rétegek absztraktizálva vannak, így a kommunikáció helyessége garantálva van. A megfelelő implementációt a *Dependency Injection* tervezési minta segítségével injektálja a *.NET keretrendszer*.

2.2.2. Adatmodell

A szerveren belül két különböző reprezentációja jelenik meg az adatoknak. A *DTO*-k a kliens és a szerver közötti kommunikációban használandóak a *Controllers* modul által. A *Models* tükrözi az adatbázist (lásd a 4. ábrát), ezt a reprezentációt részletezi ez a fejezet.

A *User* osztályban található a felhasználó teljes neve, továbbá a bejelentkezéshez szükséges adatok is itt vannak tárolva (e-mail, felhasználónév, illetve jelszó). A *ContactInfo* osztályon belül minden felhasználóról tetszőleges számú elérhetőséget lehet tárolni. Mindez a címke beolvasáskor másoknak megjelennek, ezek lehetnek e-mail, telefonszám vagy lakcím, típusokra felosztva (e-mail, lakcím vagy telefonszám). A fentebb említett információk címke beolvasáskor mind megjelennek.

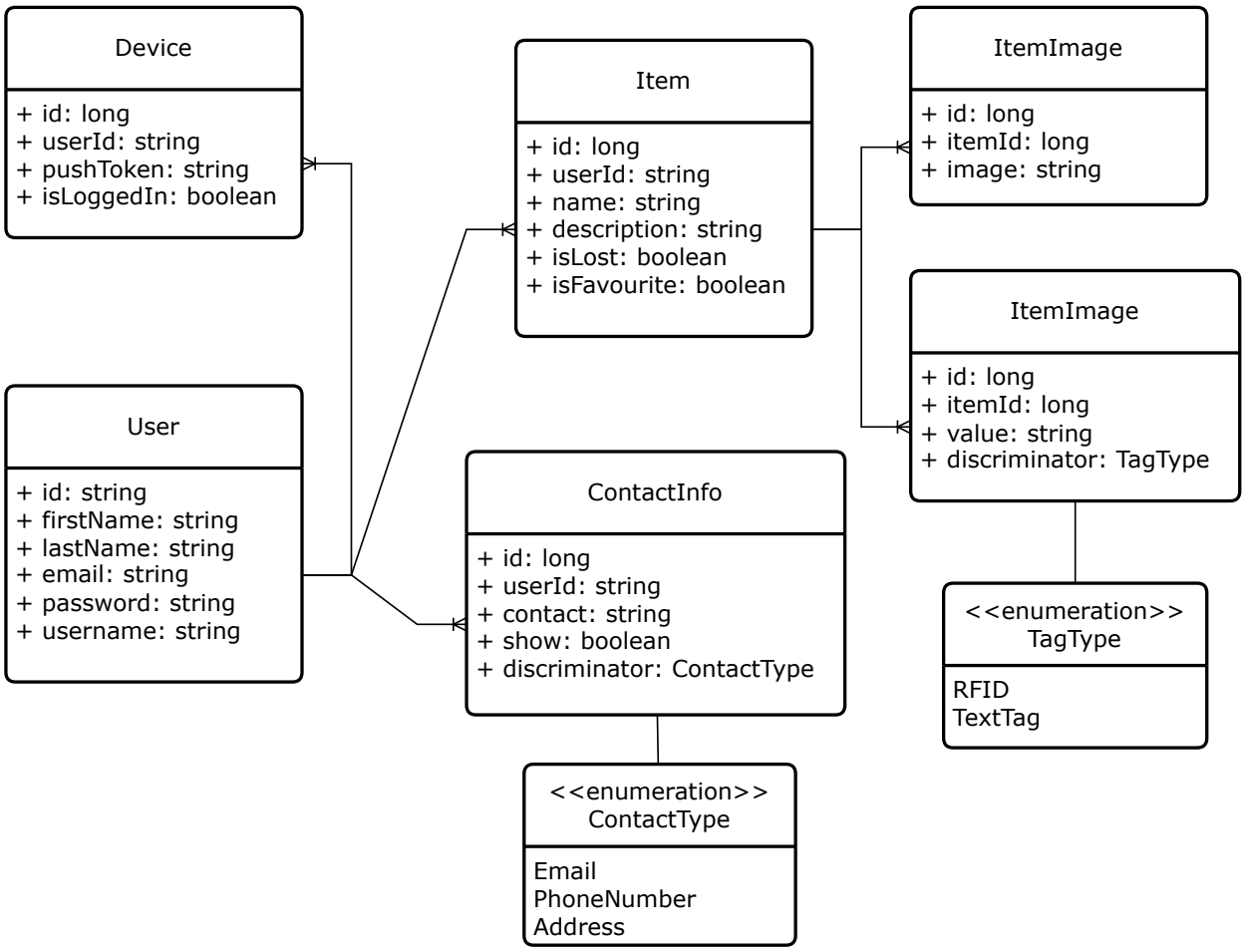
Az *Item* osztályon belül egy felhasználóhoz tartozó tárgy van eltárolva, amihez a következő információk szükségesek: név, leírás, illetve az, hogy a tárgy elveszett-e és hogy kedvenc vagy sem. Minden Item-hez tetszőleges számú képet lehet menteni az *ItemImage* osztály segítségével, illetve tetszőleges számú címkét a *Tag* osztály segítségével. A *Tag* osztályban a címke és annak típusa mentődnek le.

Minden felhasználóhoz továbbá bejelentkezéskor elmentődik egy *Device* példány, amiben megtalálható az eszköz *Push Token*-je és az, hogy a felhasználó be van-e jelentkezve ezen az eszközön. Ez az osztály az értesítések eljuttatásakor játszik fontos szerepet, mivel így minden olyan eszközre eljutnak az értesítések egyidejűleg, melyekre a felhasználó be van jelentkezve.

2.2.3. Biztonság

Mivel az applikáció különbséget tesz vendég és regisztrált felhasználó között, fontos, hogy a regisztrált felhasználók adataikhoz más ne férjen hozzá.

Ebből az okból kifolyólag a regisztrációkor megadott jelszó szerveroldalon is validálva van, 1.1. fejezetben említett jelszó komplexitás a szerver oldalon is ellenőrizve van, így a jelszavakat sokkal nehezebb programok segítségével kitalálni, amelyek minden lehetséges



4. ábra. Az alkalmazásban használt adatstruktúrák

jelszót ellenőriznek. Érvényes adatok esetén az *Identity Framework* (lásd a 3.2. fejezetet) regisztrálja a felhasználót és hash-eli annak a jelszavát, amennyiben érvénytelen adat jut a szerverhez **400 Bad Request** státusz kódot fog visszatéríteni.

A bejelentkezéskor megadott adatokat hasonlóképpen az *Identity Framework* validálja, amennyiben érvényesek az adatok a szerver oldalon egy *JSON Web Token* jön létre, amelyet visszaküld a kliensnek. A *token*-ben a felhasználó ID-ja, egy dátum amíg érvényes a token, az érvényes kiállító és egy titkos kulcs kerül mentésre. Érvénytelen adat esetén **401 Unauthorized** státusz kódot fog visszaküldeni.

A fentebb említett token ezek után minden *HTTP* kérésben szerepelni fog egy *Header*-ben. Minden olyan útvonalon, ami bejelentkezett felhasználót igényel a token tartalma validálva lesz. Amennyiben a token hiányzik vagy érvénytelen adatot tartalmaz (lejárt az érvényességi idő, helytelen a kiállító) a szerver **401 Unauthorized** státusz kódot fog visszaküldeni. Továbbá, amennyiben egy felhasználó olyan adatokhoz próbál hozzáférni amelyek nem hozzá tartoznak, a szerver **403 Forbidden**-al fog válaszolni.

3. Technológiák és eszközök

A telefonos applikáció TypeScriptben, React Native könyvtár használatával, míg a webes kliens TypeScriptben, React könyvtár segítségével íródott. A szerver C#-ban, ASP.NET Core keretrendszer felhasználásával valósult meg. Ebben a fejezetben bemutatásra kerülnek a felhasznált technológiák és eszközök.

3.1. A kliensoldali technológiák

React

A React egy komponens alapú JavaScript könyvtár, amely lehetővé teszi a felhasználói felületek hatékony frissítését [8]. A komponensek előnye az újrahasználhatóság, amely elősegíti az alkalmazások hatékony fejlesztését. Ezen komponenseket és azok attribútumát a VDOM (Virtual Document Object Model) tartalmazza. A VDOM belső állapotának változásakor összehasonlítja a régi és az új VDOM-ot, amiből meghatározza a változott elemeket. Csak a változott elemeket frissíti a valódi DOM-on, amely a frissítéshez szükséges műveletek számát minimalizálja.

React alkalmazások fejlesztése történhet JavaScriptben is, viszont a JSX (JavaScript eXtension) használata jobb olvashatóságot kínál. A JSX egy JavaScript kiterjesztés, amely segítségével a React komponenseket egy XML szerű szintaxis használatával definiáljunk. A JSX kódot egy előfeldolgozó JavaScriptre fordítja, melyet a DOM már értelmezni tud. A JSX-nek egy másik előnye az, hogy attribútumokon keresztül bármilyen típus átadható, így a komponensek közti kommunikáció sokkal egyszerűbb.

React Native

A React Native egy nyílt forráskódú keretrendszer, amely React segítségével lehetővé teszi a natív mobilalkalmazások fejlesztését Android és iOS platformokra [7]. A JavaScript kód feldolgozásáért és futtatásáért a JSC (JavaScript Core) felelős, amely egy JIT (Just-In-Time) kompilátorral rendelkezik. Futtatás előtt a JavaScript kódot a JIT gépi kóddá fordítja, ezért a futtatás sebessége gyorsabb más értelmezőkkel szemben. Emellett a JSC támogatja a TypeScriptet és lehetővé teszi a kód módosítását, anélkül, hogy a teljes alkalmazást újra kéne fordítania az adott platformra. Ennek köszönhetően a fejlesztési folyamat is gyorsabb.

Egy React Native alkalmazás JavaScript kódból és natív komponensekből áll. A JavaScript

kód a React Native által biztosított API-on keresztül kommunikál a platform specifikus komponensekkel. A natív komponensek platform specifikusak, melyeket a JavaScript kód használ a megjelenítéshez. Ez lehetővé teszi, hogy az alkalmazás egyetlen kódbázisból épüljön fel és natív platform funkciókat is kihasználhasson.

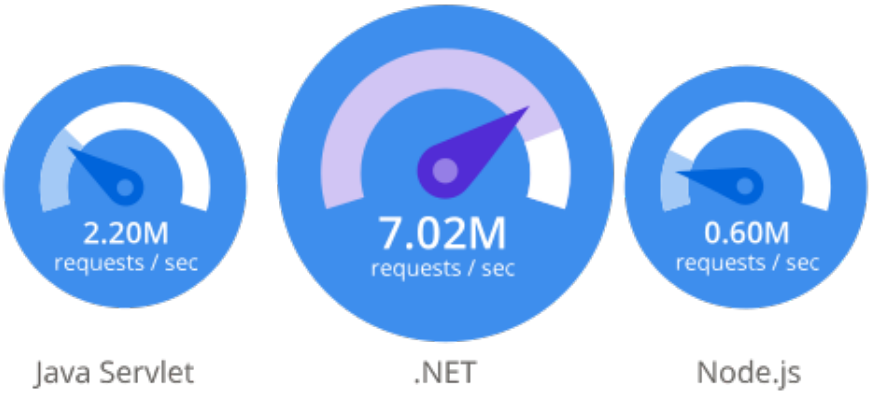
Expo

Az Expo egy nyílt forráskódú keretrendszer, amely segítségével React Native alkalmazásokat lehet fejleszteni [4]. Számos funkcióval és eszközzel rendelkezik, melyek az alkalmazások fejlesztését, tesztelését és publikálását könnyítik meg. Egy CLI (Command Line Interface) használatával könnyen létre lehet hozni egy új projektet, amely automatikusan konfigurálja a React Native alkalmazást beépített Expo szolgáltatásokkal. Számos beépített szolgáltatással és API-val rendelkezik, mint például hozzáférés a kamerához, push értesítések és helymeghatározás. Mindemellett rendelkezik egy Expo Client alkalmazással, amely segítségével valós eszközön tesztelhetőek az alkalmazások anélkül, hogy ki lennének telepítve a Google Play vagy App Store áruházakba. Az Expo saját szerverén keresztül lehetővé teszi az alkalmazások publikálását. Az alkalmazás az Expo CLI segítségével publikálható és rendszer által generált publikus linken keresztül elérhetővé válik az Expo Client-ből.

Typescript

A TypeScript egy nyelvi kiterjesztése a JavaScriptnek, amely lehetővé teszi a statikus típusellenőrzést [9]. Megengedi a változók és paraméterek típusának deklarálását, amely nagy szerepet játszik fejlesztés során az esetleges hibák beazonosításában. Emellett használhatók interfészek, osztályok és felsorolások, melyek használatával strukturáltabb kód írható. Futtatás során a TypeScript *transpilere* a nyelvi szabályok ellenőrzése után egyszerű JavaScript kódra fordítja a forráskódot.

Mivel a TypeScript kódot ugyanaz a JavaScript motor futtatja, ezért JavaScript kód is futtatható TypeScript környezetben. A nyelv széles körben elterjedt annak köszönhetően, hogy a legtöbb fejlesztési könyvtár használatát támogatja, mint például a React, Angular, VueJS. Ezen könyvtárak használatához külön konfigurációra van szükség, melyet a tsconfig nevű fájl tartalmaz. Ez mindazon típusdefiníciókat foglalja magába, melyek leírják a könyvtár szerkezetét a *transpiler* számára.



5. ábra. A .NET teljesítménye Java Servletek-hez és NodeJS-hez hasonlítva

3.2. A szerveroldali technológiák

ASP.NET Core

Az *ASP.NET Core* egy cross-platform, nyílt forráskódú keretrendszer webalkalmazások fejlesztésére, melynek alapja a .NET Core. A Microsoft hivatalos terméke és teljes támogatást élvez, mégis ingyenes [16]. Az ASP.NET gyorsabb, mint bármely népszerű web keretrendszer a TechEmpower felmérése szerint. A 2022-es felmérés eredményeiből [17] kiemelve látható a .NET, Java Servletek és NodeJS összehasonlítása az 5.[4] ábrán, amely a Microsoft által vallott teljesítményt illusztrálja.

Identity Framework

Az *Identity Framework* az ASP.NET Core egyik nyílt forráskódú keretrendszere, amely az autentikációs és autorizációs folyamatok megkönnyítésére lett létrehozva. Számos lehetőséget támogat a bejelentkezésre: a megszokott felhasználónév és jelszó kombinációval, két lépcsős hitelesítés, illetve külső fél általi bejelentkezés. Ezen túl még sok más funkcionalitást biztosít, például szerepkörök vagy e-mail cím megerősítés.[13].

[4] `https://dotnet.microsoft.com/static/images/redesign/shared/tech-empower-results.svg` utolsó megtekintés dátuma: 2023.04.27

Entity Framework

Az *Entity Framework* egy nyílt forráskódú, cross-platform ORM (*Object-Relational Mapping*) keretrendszer, amely lehetőséget biztosít a fejlesztőknek, hogy objektumokon keresztül kommunikáljanak egy adatbázissal alacsony szintű SQL írása nélkül. Számos relációs adatbázist támogat: MySQL, SQLServer, PostgreSQL [2].

A kommunikáció egy kontextus objektumra alapszik, illetve modellekre, melyek az adatbázisban levő táblákat tükrözik. Támogatja a *Code First* (az adatbázis a modellek alapján lesz létrehozva) illetve a *Database First* (a modellek az adatbázis oszlopai szerint lesznek létrehozva) stílusokat, a fejlesztőre bízva melyik oldal megírása kényelmesebb.

PostgreSQL

A *PostgreSQL* egy nyílt forráskódú, relációs adatbázis-kezelő rendszer, amely az SQL nyelvet kombinálja különböző funkciókkal, hogy biztonságosan eltárolja a legkomplexebb adatokat is [15]. Népszerűsége az architektúrája, megbízhatósága, robusztussága és bővíthetőségének eredménye. Lehetőséget ad saját adattípusok definiálására, illetve más programozási nyelvek használatára, függvények és eljárások megírására [6].

Az adatokat táblákban tárolja el, ahol az oszlopok az entitás attribútumait tartalmazzák (*pl.: név, telefonszám stb.*.), a sorokban pedig egy entitás adatai találhatóak.

3.3. Eszközök

Git

A *Git* egy osztott verziókövető rendszer, amely lehetőséget nyújt változtatások számontartására és a párhuzamos fejlesztésre. A Git más verziókövetőkkel (Subversion, CSV, Perforce) ellentétben, nem csak a változtatásokat tárolja el a verziók között, hanem a teljes fájlrendszert. Amennyiben egy fájlon nem történik változtatás, akkor csak egy mutatót ment el az előző verzióban szereplő fájlra. Továbbá, a Git rendelkezik egy *Staging Area*-val is, ahol a változtatásokat ellenőrizni lehet már azelőtt is, hogy commit jöjjön létre.

GitLab

A *GitLab* egy nyílt forráskodú projektmenedzsment felület, ahol a Git alapú tárolókat kezelni lehet. Továbbá, az oldalon minden szükséges eszköz megtalálható, ami egy projekt

DevOps szükségleteit kielégíti. Ezen túl, lehetőséget biztosít taszkmenedzsmentre, forráskód ellenőrzésre és folyamatos integráció és folyamatos kitelepítésre [12].

A *DevOps* az informatikai fejlesztés (*development*) és működés (*operations*) összekapcsolásával növeli a fejlesztés és kiadás hatékonyságát, sebességét és biztonságát a hagyományos módszerekhez képest.

Docker

A *Docker* egy fejlesztésre, publikálásra, kitelepítésére és legfőképp virtualizációs eszközként használandó felület. Lehetőséget nyújt arra, hogy az applikációkat elkülönítve futtassuk, teljesen függetlenné téve, ezáltal azonos futási környezetet használhatunk, függetlenül attól, hogy milyen eszközön vagyunk [1]. Mindennek kulcsa a *container*, amely egy virtuális környezetet hoz létre és azon belül eltárolja az alkalmazást és minden függőségét, majd ott futtatja is. Ezek a virtuális környezetek kihasználják az eszközön levő Linux kernelt, így sokkal nagyobb gyorsaságot érhetnek el, mint egy virtuális gép.

A *Docker-Compose* egy eszköz, aminek segítségével több container-t lehet futtatni párhuzamosan, így skálázható és moduláris rendszerek jönnek létre. Ennek segítségével megoldható, hogy egy container-ben fusson a webalkalmazás, míg a másikban a Postgres adatbázis és mindez egyszerre, egy parancsra kitelepítésre kerüljön.

Függőségkezelés

A kliens- és szerveroldalon is különböző eszközökkel van megoldva a függőségek kezelése, nagy mértékben megkönnyítve ezek telepítését, törlését és verziókövetését.

A mobil applikáció és a weboldal is *npm*-t használ, mint függőségkezelő. A hivatalos tájékoztató szerint [11], "az npm a világ legnagyobb szoftver nyilvántartója". A weboldalon több mint kétmillió Javascript csomag található meg, amit tetszőlegesen bármikor le lehet tölteni. Továbbá, egy Command Line Interface-t (parancssoros felhasználói felület) is biztosít, a fejlesztők általában ennek segítségével kommunikálnak az npm-el és tudnak letölteni, törölni és frissíteni csomagokat.

A szerveroldalon a .NET és .NET Core hivatalos függőségkezelője van alkalmazva, a *NuGet* [14]. A Microsoft által támogatott felületen a fejlesztők tetszés szerint használhatják fel a már meglévő könyvtárakat, illetve hozhatnak létre és publikálhatnak saját könyvtárt is. Rendelkezik egy CLI-al is, illetve a Microsoft hivatalos .NET fejlesztési környezetébe (*Visual Studio*) is be

van építve, saját grafikus interfésszel rendelkezik, ahol böngészni, letölteni lehet csomagokat és a már letöltött csomagokat kezelni is.

Statikus kódellenőrzők

A statikus kódellenőrzők szerepe a projektben, a kód minőségének ellenőrzésé anélkül, hogy azt ténylegesen futtatva legyen. Általában az adott programozási nyelv szintaktikai és stilúsbeli hibáit vizsgálják. Ezek az eszközök nagyban megkönnyítik a fejlesztők dolgát, mivel képesek előre jelezni olyan hibákat amelyek csak később kerülnének felszínre, így azokat jelentősen gyorsabban és hatékonyabban javítani lehet.

A projekt keretén belül az *ESLint* [3] volt alkalmazva a weboldal és mobil applikációban is. Az *ESLint* egy népszerű, nyílt forráskódú statikus kód ellenőrző, amelyet a *Javascript* programozási nyelvhez fejlesztettek ki. Több típusú ellenőrzési szabállyal lehet használni, amit a fejlesztő kibővíthet saját szabályokkal, így teljesen személyre szabható funkciókkal ellátva. Alap beállításokban az *ESLint* típusbiztonsági, stílus és szintaxis hibákat képes lekezelni. Továbbá, beépíthető az integrációs folyamatba, így automatikusan ellenőrizheti a kód minőségét.

4. Munkamódszerek

4.1. Scrum

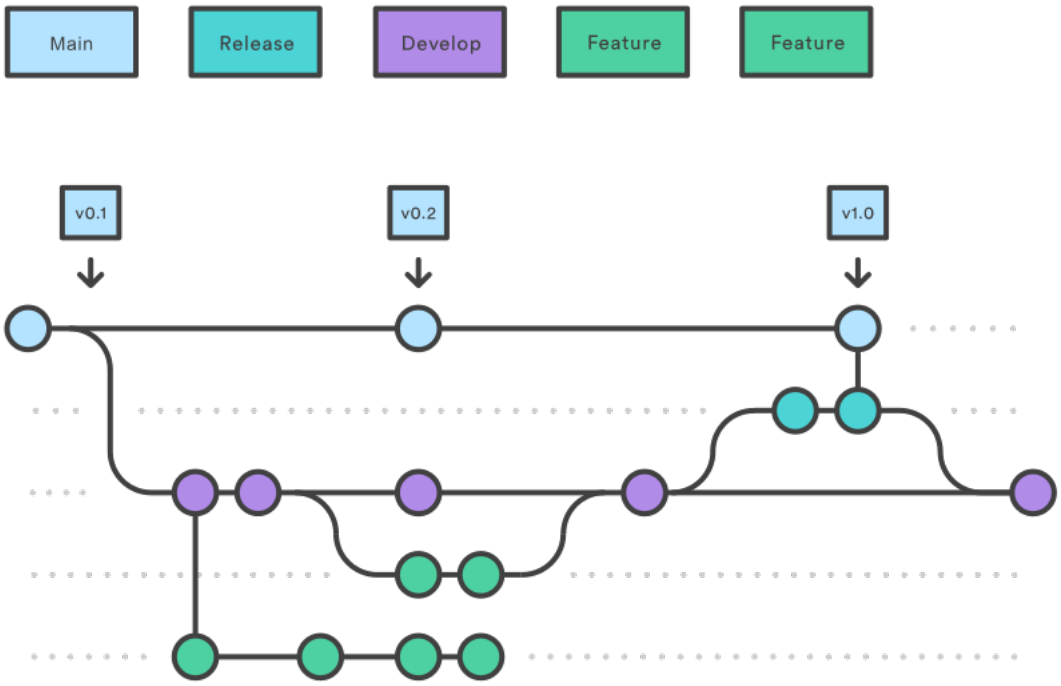
A *Scrum* egy agilis projektmenedzsment keretrendszer, amely egy rugalmas és iteratív munkafolyamatot alkalmaz, amely lehetőséget nyújt a fejlesztőknek, hogy gyorsabban és hatékonyabban tudjanak összedolgozni. A fejlesztés *sprint*-ekbe osztódik fel, amelyek hossza általában kettő vagy három hét. Minden sprint elején egy *planning* alatt a csapat a szükséges funkcionalitásokat felméri relevancia és komplexitás szerint. Ezek után a *sprint backlog*-ba kerülnek azok a teendők, amelyekről a csapat úgy véli, hogy a sprint végéig el lehet végezni. A sprint során napi megbeszélések vannak tartva, ahol mindenki röviden elmondja, hogy mennyit haladott, milyen problémákba ütközött és mivel fog foglalkozni aznap. A sprint végén két fontos gyűlés is található, az egyike a *demo*, ahol sor kerül az elkészült funkcionalitások bemutatására. A másik fontos megbeszélés a *retrospective*, ahol a csapattagok megoszthatják egymással a sprint-tel kapcsolatos pozitív és negatív tapasztalokat, illetve javaslatokat tehetnek arra, hogy mi kéne előnyt élvezzen a következő sprint-ben és hogyan kerülhetőek el az említett problémák.

4.2. GitLab Issues

A *GitLab Issues* a teendők osztályozására használható felület, amit a 3.3. fejezetben említett *GitLab* biztosít a hatékonyabb fejlesztés érdekében. A grafikus felületen több oszlop található, melyek a feladatok állapotát jelzik. A 4.1. alfejezetben említett *Sprint Backlog* tartalmazza azokat a teendőket, melyeket a jelenlegi sprint alatt kell befejezni. Mindenki számára elérhetőek ezek a feladatok, kiválaszhatja tetszés szerint melyiken dolgozna, és azt az *In Progress* oszlopba mozdítja, jelezve a csapatnak, hogy elkezdett dolgozni rajta. A *Waiting* egy ritkábban használt oszlop, ide kerülnek azok a teendők, amelyek valamelyik más feladattól függenek, infrastruktúra vagy valamilyen külső tényező miatt még nem lehet rajtuk dolgozni. A *Review* oszlop tartalmazza azokat a feladatokat, amelyek be vannak fejezve és ellenőrzésre várnak valakitől, illetve a *Done* oszlop azokat, amelyek el lettek fogadva.

4.3. Gitflow

A *Gitflow* egy elágazási stratégia (*branching model*) a Git verziókövetőn belül, amely biztosítja az átlátható munkát és stabil kód kitelepítését. A GitFlow szerint, két állandó



6. ábra. Gitflow illusztráció

elágazásnak kell léteznie, a *main* vagy *master* és a *develop* vagy *dev*. A *main* ág mindig stabil, kitelepítésre készen álló kódot tartalmaz, általában fontosabb verziók kerülnek ide be. A *dev* a fejlesztési folyamaton belül a fő elágazás, minden más *branch* ebből indul ki és ide is lesz vissza *merge*-lve. A fejlesztés alatt minden feladatnak saját *branch*-je van, aminek a neve elején található egy kulcsszó, ami jelzi a feladat típusát (*feature, fix,* stb..) és utána található a funkcionalitás neve. A 6. ábrán található a *Gitflow* stratégia egy szemléltetése is. [5]

4.4. Folyamatos integráció és kitelepítés

A folyamatos integráció (*Continuous Integration*) fontos szerepet játszik a projekt megvalósításában, lehetőséget biztosít arra, hogy automatikus ellenőrző lépések fussanak le minden változtatás után. Mindez a GitLab CI/CD [5] által történik, ahol úgynevezett *pipeline*-okat lehet létrehozni, melyek lépésekre bontott feladatokat végeznek el. A *pipeline*-okban definiált feladatok jól meghatározott események bekövetkezésekor futnak le. Például az 1. kódrészletben látható *check* taszk akkor fog lefutni, amikor bármely branch-en változtatás történik és ellenőrzi, hogy a kód szintaktikailag helyes, de a *prepare-deploy* taszk kizárólag a *develop* ágon fut le, amikor oda egy új funkcionalitás kerül be, előkészítve a

[5]Forrás: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-work, utolsó megtekintés dátuma: 2023-04-20

```yaml
stages:
  - build
  - prepare-deploy
  - deploy

check:
  stage: build
  image: mcr.microsoft.com/dotnet/sdk:latest
  script:
    - 'dotnet restore'
    - 'dotnet build --no-restore'

build-image:
  stage: prepare-deploy
  image: docker/compose:latest
  services: [ docker:dind ]
  only:
    - develop
  # ...
```

1. kódrészlet. Részlet az alkalmazás szerver *pipeline* konfigurációs állományból

kitelepítéshez szükséges *docker image*-t.

A folyamatos integrációt követi a folyamatos kitelepítés (*Continous Delivery*, amely a szerver és a weboldal kitelepítését fogja automatizálni. Amiután a *develop* ágon a *docker image* elkészül az felkerül a *GitLab Container Registry*-be, majd ez lesz felhasználva a kitelepítés során.

A szerver és a weboldal külön *docker container*-ekben vannak kitelepítve. A szerver és az adatbázis saját virtuális hálozaton kommunikál (lásd a 2. kódrészletet), ami a megfelelő *docker compose* állományban van létrehozva, míg a szerver és a weboldal között HTTP kérések segítségével történik a kommunikáció. Annak ellenére, hogy külön *docker image*-ként vannak kitelepítve, a szerver és a weboldal ugyanazon a címen érhető el. Ezek között a *traefik* oldja meg a kérések továbbirányítását, figyelembe véve, hogy a kérés elején szerepel a */api/* részlet vagy sem, amennyiben igen a kérés az alkalmazás szerverhez fog kerülni, különben a weboldalhoz.

A *traefik* egy nyílt forráskódú, dinamikus *reverse proxy* és *load balancing* szoftver, amelyet konténerizált környezetekben alkalmaznak. Lehetővé teszi, hogy az alkalmazások könnyen kommunikáljanak egymással és a külvilággal, miközben automatikusan kezeli az útvonalakat és a terhelést is.

```
services:

  backend:
    depends_on:
      - myowner_db
    networks:
      - proxy
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.myowner.rule=Host(`myowner.test.edu.codespring.ro`) &&
    # ...


  myowner_db:
    container_name: 'myowner_db'
    image: postgres:14
    networks:
      - proxy
    # ...
```

2. kódrészlet. Részlet az alkalmazás szerver *docker compose* állományából

4.5. Kódellenőrzés

A kódellenőrzés nagy szerepet játszik a kódbázis minőségében és esetleges hibák felderítésében. Két fajta kódellenőrzés létezik: automatikus és manuális.

Automatikus ellenőrzést a 3.3 fejezetben említett ESLint kínál frontenden, amely biztosítja, hogy a kód jól meghatározott szabályoknak eleget tegyen azáltal, hogy beazonosítja a hibákat.

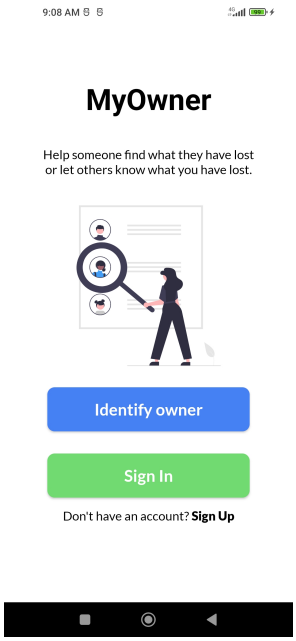
A GitLab verziókövető rendszer lehetőséget nyújt kézi kódellenőrzésre, melynek lényege a logikai hibák kiszűrése, amiket nem lehet automatikusan detektálni. Különböző branch-ek visszacsatolása más ágakba *merge request*eken keresztül valósul meg. A *merge request* lehetőséget nyújt a változtatások átnézésére és esetleges hibák jelzésére megjegyzések hozzáadása által. A projekt esetében a hibákra a megfelelő mentorok hívják fel a figyelmet azáltal, hogy átnézik *merge requesteket*. A javított kódot a felülvizsgáló újra átnézi, melyet jóváhagy vagy új megjegyzésekkel egészít ki. Az összes megjegyzés tisztázása után a változtatások bekerülnek a cél branch-be. Egymás munkájának az ellenőrzése biztosítja a kód stabilitását és minőséget.

5. A MyOwner applikáció működése

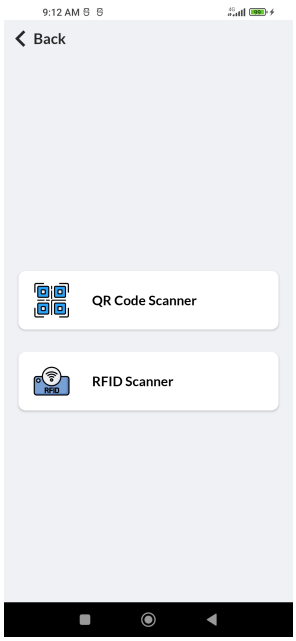
Az applikáció megnyitásakor a *Welcome screen* fogadja a felhasználót, ahonnan a 7a. ábrán látható 3 funkcionalitás érhető el. Az első gomb a szkenner kiválasztásához irányit át, a második pedig a bejelentkezéshez. A második gomb alatti vastagított szöveg a regisztrációhoz irányít át.

A szkenner funkcionalitás az egyik legfontosabb funkció, melynek segítségével a tárgyak tulajdonosait tudjuk beazonosítani. Két fajta típusú szkenner érhető el, egy QR kód és egy RFID tag olvasó, melyeket a 7b. ábrán láthatunk. A QR kód olvasó működéséhez először engedélyezni kell az alkalmazás hozzáférését a készülék kamerájához, ami után elérhető lesz a hátsó és az elülső kamera is, ha létezik. Ahhoz, hogy az RFID olvasó működjön a készüléknek rendelkeznie kell NFC technológiával (lásd a 7c. ábrát). Ez esetben az RFID taget a készülék hátuljához kell tartani, amíg sikeresen leolvasódig.

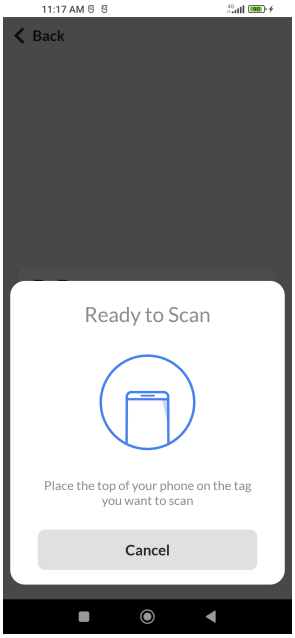
Sikeres szkennelés esetén a felhasználót átirányítja a tárgy információihoz, melyet a 8a. ábrán láthatunk. Megjeleníti a tárgy tulajdonosát, annak elérhetőségeit, valamint esetenként képet és leírást a tárgyról. Abban az esetben, ha a MyOwner applikáció nélkül történt a szkennelés, akkor átirányítja a felhasználót a webes kliensre (lásd a 8b. ábrát), amely reszponzív, vagyis különböző nagyságú képernyőkön is kényelmesen használható. Innen számos opció érhető el, mint például tárgy információinak megtekintése, telefoos applikáció megnyitása, webes QR kód szkennelő, valamint tárgy keresése szöveges címke szerint.



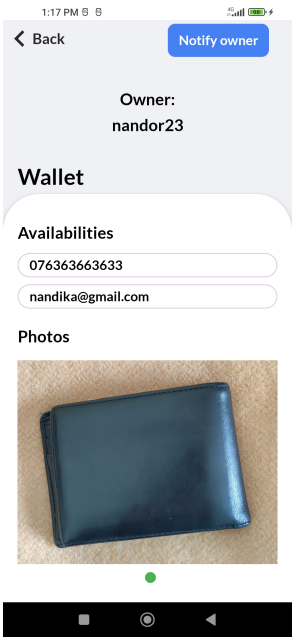
(a) Welcome screen - A felhasználót fogadó képernyő



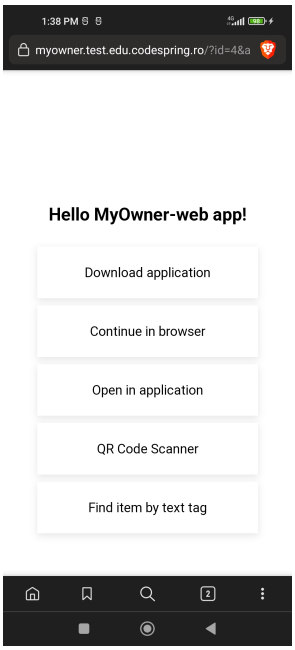
(b) Scanners screen - A szkenner típusának kiválasztása



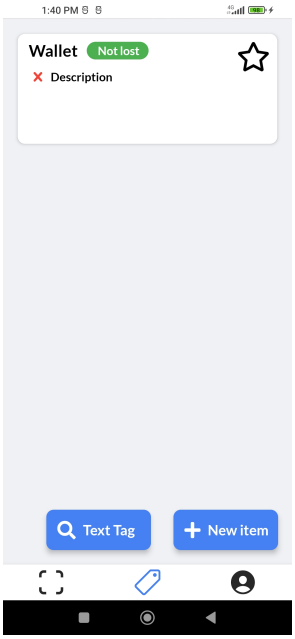
(c) RFID olvasó



(a) A tárgy információt megjelenítő képernyő



(b) Webes átirányítás applikáció nélküli szkennelés esetén

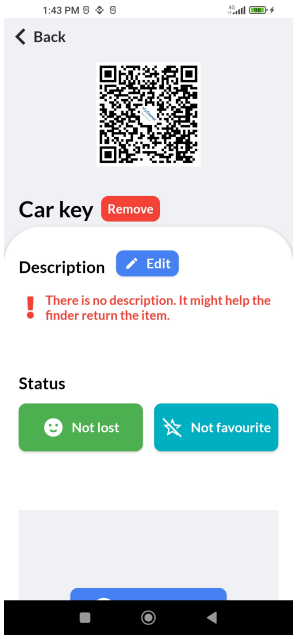


(c) Felhasználó tárgyainak listája

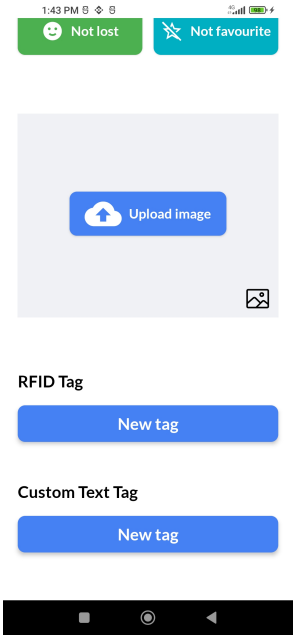
Regisztráció esetén meg kell adni a nevet, felhasználó nevet, e-mail címet és jelszót. A jelszónak biztonsági okok miatt kötelezően tartalmaznia kell egy nagy betűt, számot és speciális karaktert.

Bejelentkezés után a felhasználót saját hozzáadott tárgyainak oldala fogadja, melyet a 8c. ábrán láthatunk. A képernyő legallján a navigációs sáv található, melyen 3 ikon helyezkedik el. A kék színű ikon mindig azt az oldalt jelöli, amelyen a felhasználó jelenleg tartozkodik. Az első ikon a szkenneléshez, a második a tárgyak listájához, az utolsó pedig a felhasználó információihoz irányít át. A navigációs sáv felett 2 gomb található, az elsővel szöveges címke szerint azonosíthatunk tárgyakat, a másodikkal új tárgyat vezethetünk be. Új tárgy hozzáadása esetén 2 szövegdoboz van, egy a tárgy nevének és a másik annak leírásának, viszont ezek közül csak a tárgy nevét kell megadni, mivel a leírás megadása opcionális.

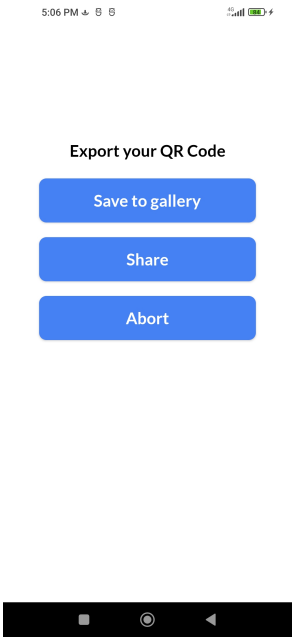
A tárgyak listájából egyet kiválasztva átirányítja a felhasználót a tárgy információinak oldalára (lásd a 9a. és 9b. ábrát). Az oldal felső részén látható a QR kód, mely automatikusan generálódik. Ennek elérintésével a QR kód elmenthető kép formájában vagy megosztható más alkalmazásokon, mint például Messenger, Email, Facebook (lásd a 9c ábrát). A tárgy neve alatt, a *Description* résznél található a tárgy leírása. Egy piros szöveg figyelmezteti a felhasználót abban az esetben ha nincs leírás megadva, mivel ez esetenként segíthet fontos információt nyújtani a tárgy megtalálójának. A *Status* szekcióban az első gombbal beállítható az, hogy a tárgy elveszett vagy sem, amely a tárgyak listája közötti navigálásban segít, amikor egy



(a) Egy tárgy információinak oldala



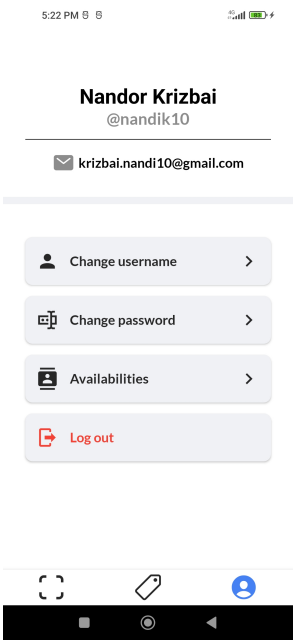
(b) Egy tárgy információinak oldala



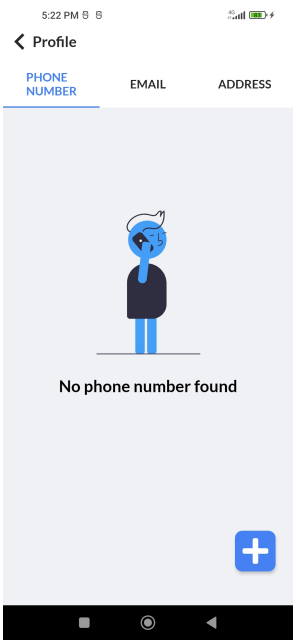
(c) QR kód exportálása

bizonyos tárgyat keresünk. A második gombbal kedvencnek jelölhetőek a tárgyak, így a tárgyak listáján a kedvencnek jelölt tárgyak lesznek elsőként megjelenítve. Az *Upload Image* gombbal képeket tölthetünk fel a tárgyról, amely FullHD minőségnél nagyobb képeket átméretez FullHD-ra, így kevesebb tárhelyet foglalnak az adatbázisban. A képek feltöltése alatt 2 gomb látható, egy RFID és egy szöveges címke hozzáadására. Az RFID-t csak le kell olvasni, a Text tag-nél pedig egy általunk választott szöveget kell megadnunk. A szöveghez automatikusan egy véletlenszerűen generált szöveg fűződik hozzá, ami az adatbányászat elkerülésére szolgál. Ezt a címkét meg lehet osztani más alkalmazásokon keresztül is: kinyomtatható vagy kézzel felírható a tárgyra. Megeshet, hogy a megtaláló készülékének nincs kamerája, így a weboldalon vagy akár az applikációban is beazonosíthatja a tárgy tulajdonosát a szöveges címke beírásával.

A profil menüpontban a név, felhasználónév és e-mail cím van feltüntetve (lásd a 10a. ábrát). Emellett 4 gomb található, az elsővel felhasználó nevet, a másodikkal pedig jelszót lehet cserélni. A harmadik gomb az elérhetőségek listájához irányít át, melyet a 10b. ábrán láthatunk. A *Back* gomb alatt 3 szekció van, ahol a felhasználó elérhetőségei csoportosítva vannak telefonszám, e-mail és lakcím szerint. A képernyő alján egy + gomb segítségével új elérhetőséget lehet hozzáadni, ahol az elérhetőség típusát kell kiválasztani, egy szövegdobozt kell kitölteni és egy checkboxot lehet bejelölni. A checkbox bejelölésével az adott elérhetőség megjelenik, ha valaki beazonosítja valamely elveszett tárgyunkat, így bármely elérhetőség láthatósága kényelmesen módosítható. A *Log out* gombbal a felhasználó kijelentkezhet



(a) Profil információinak oldala



(b) Elérhetőségek megtekintése

fiókjából.

6. Következtetések és továbbfejlesztési lehetőségek

A projekt fejlesztése során kialakult egy három komponensből álló szoftverrendszer, amelyek használatával sokkal egyszerűbb megtalálni egy elveszett tárgy tulajdonosát.

A mobil applikációban mindenki bevezetheti és kezelheti a saját tárgyait és elérhetőségeit, különböző címkéket adhatnak hozzá a tárgyaikhoz. Az applikáció segítségével a címkéket be lehet olvasni és a megjelenített elérhetőségek segítségével könnyeben el lehet érni a tulajdonos. Továbbá, értesítést lehet küldeni a tulajdonosnak, jelezve neki arról, hogy valaki megtalálta a tárgyát.

A weboldal segítségével megpróbálunk azon megtalálókon segíteni, akiknek nem áll rendelkezésükre a mobil applikáció, a weboldal segítségével QR-kódot és a szöveges kódot is be lehet olvasni és hasonlóképpen megjelennek az információk, mint a mobil applikáción.

Továbbfejlesztési lehetőségként számos opció merül fel:

- Növelni a támogatott címkék számát, hogy a lehető legtöbb típusú eszközt lehessen használni az applikáció keretén belül
- Egy gamification jellegű rendszer bevezetése, amely díjazná azon felhasználókat, akik a lehető legtöbb tárgyat juttatnak vissza a tulajdonosukhoz
- Egy *live chat* funkció, aminek segítségével tovább könnyítenénk a kommunikációt a megtaláló és tulajdonos között.
- A felhasználó profiljához több információ megadása, első prioritásként egy profilkép, hogy a megtaláló a lehető legkönnyebben tudjon megbizonyosodni róla, hogy a megfelelő emberhez juttatja el a tárgyat.

Hivatkozások

[1] Hivatalos Docker dokumentáció. *Docker: Get started*. URL: https://docs.docker.com/get-started/ (elérés dátuma: 2023. ápr. 22.)

[2] Hivatalos Entity Framework dokumentáció. *Entity Framework documentation*. URL: https://learn.microsoft.com/en-us/ef/ (elérés dátuma: 2023. ápr. 14.)

[3] Hivatalos ESLint dokumentáció. *Documentation - ESLint*. URL: https://eslint.org/docs/latest/ (elérés dátuma: 2023. ápr. 22.)

[4] Hivatalos Expo dokumentáció. *Core concepts*. URL: https://docs.expo.dev/core-concepts (elérés dátuma: 2023. ápr. 20.)

[5] Hivatalos Gitlab CI/CD dokumentáció. *Gitlab CI/CD*. URL: https://docs.gitlab.com/ee/ci/ (elérés dátuma: 2023. ápr. 22.)

[6] Hivatalos PostgreSQL dokumentáció. *PostgreSQL: Documentation, Chapter 42. Procedural Languages*. URL: https://www.postgresql.org/docs/current/xplang.html (elérés dátuma: 2023. ápr. 14.)

[7] Hivatalos React Native dokumentáció. *Core Components and Native Components*. URL: https://reactnative.dev/docs/intro-react-native-components (elérés dátuma: 2023. ápr. 15.)

[8] Hivatalos ReactJS dokumentáció. *Tutorial: Tic-Tac-Toe*. URL: https://react.dev/learn/tutorial-tic-tac-toe (elérés dátuma: 2023. ápr. 15.)

[9] Hivatalos TypeScript dokumentáció. *TypeScript for the New Programmer*. URL: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html (elérés dátuma: 2023. ápr. 20.)

[10] Cloud Computing-al kapcsolatos fogalmak. *What is a RESTful API??* URL: https://aws.amazon.com/what-is/restful-api (elérés dátuma: 2023. ápr. 13.)

[11] Hivatalos npm tájékoztató. *About npm*. URL: https://docs.npmjs.com/about-npm (elérés dátuma: 2023. ápr. 16.)

[12] Hivatalos Gitlab tájékoztató. *Why Gitlab?* URL: https://about.gitlab.com/why-gitlab/ (elérés dátuma: 2023. ápr. 14.)

[13] Hivatalos Identity Framework tájékoztató. *Introduction to Identity on ASP.NET Core*. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio (elérés dátuma: 2023. ápr. 14.)

[14] Hivatalos NuGet tájékoztató. *What is NuGet and what does it do?* URL: https://learn.microsoft.com/en-us/nuget/what-is-nuget (elérés dátuma: 2023. ápr. 16.)

[15] Hivatalos PostgreSQL tájékoztató. *PostgreSQL: About*. URL: https://www.postgresql.org/about/ (elérés dátuma: 2023. ápr. 14.)

[16] Hivatalos ASP.NET tájékoztató. *What is ASP.NET?* URL: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet (elérés dátuma: 2023. ápr. 13.)

[17] TechEmpower. *Round 21 results*. URL: https://www.techempower.com/benchmarks/#section=data-r21&hw=ph&test=plaintext (elérés dátuma: 2023. ápr. 14.)